IBM Systems - iSeries

Programming
IBM Toolbox for Java

Version 5 Release 4

IBM Systems - iSeries

Programming
IBM Toolbox for Java

Version 5 Release 4

Note

Before using this information and the product it supports, read the information in "Notices," on page 729.

Tenth Edition (February 2006)

This edition applies to version 5, release 4, modification 0 of IBM Toolbox for Java (product number 5722–JC1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

IBM Toolbox for Java

IBM® Toolbox for Java™ is a set of Java classes that allow you to use Java programs to access data on your iSeries™ servers. You can use these classes to write client/server applications, applets, and servlets that work with data on your iSeries. You can also run Java applications that use the IBM Toolbox for Java classes on the iSeries Java virtual machine (JVM).

IBM Toolbox for Java uses the iSeries Host Servers as access points to the system. Because IBM Toolbox for Java uses communication functions built into Java, you do not need to use IBM iSeries Access for Windows® to use IBM Toolbox for Java. Each server runs in a separate job on the server, and each server job sends and receives data streams on a socket connection.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 728.

What's new

This topic highlights the changes made to IBM Toolbox for Java in V5R4.

IBM Toolbox for Java is available in the following forms:

- The licensed program for IBM Toolbox for Java, 5722-JC1, Version 5 Release 4 (V5R4) installs on V5R2 and later versions of i5/OS™. From a client, IBM Toolbox for Java connects back to V5R2 and later versions of i5/OS.
- i5/OS also includes the non-graphical classes of IBM Toolbox for Java optimized for use when running IBM Toolbox for Java classes on the iSeries Java virtual machine (JVM). So if, for example, you do not have a need for the graphical functionality of the licensed program, you can still easily use IBM Toolbox for Java. For more information, see Jar files.
- IBM Toolbox for Java is also available in an open source version. You can download the code and get more information from the JTOpen Web site.

Enhancements to IBM Toolbox for Java JDBC support

For information about enhanced JDBC functions, see "Enhancements to IBM Toolbox for Java JDBC support for V5R4" on page 307. For information about new JDBC properties, see "IBM Toolbox for Java JDBC properties" on page 311.

New classes

The following classes have been added since V5R3. All classes listed here are in package "com.ibm.as400.access," unless otherwise noted.

- "BidiConversionProperties" on page 29
- "CallStackEntry" on page 29
- "IFSFileReader" on page 55
- "IFSFileWriter" on page 57
- "IFSSystemView" on page 60
- "ISeriesNetServer" on page 61
- "SaveFile" on page 41
- SignonHandler (interface)
- "Subsystem" on page 42

Enhanced classes

Significant enhancements were made to the following classes. All classes listed here are in package "com.ibm.as400.access," unless otherwise noted.

- Many of the "JDBC classes" on page 61
- "IFSFile class" on page 52
- JarMaker and AS400ToolboxJarMaker (in the "utilities" package)
- FTP and AS400FTP

Deprecated packages

The following packages have been deprecated since V5R3:

- com.ibm.as400.vaccess
- com.ibm.as400.resource

Deprecated classes

The following classes have been deprecated since V5R3:

- utilities.AS400ToolboxInstaller
- com.ibm.as400.access.NetServer has been replaced by "ISeriesNetServer" on page 61.
- com.ibm.as400.access.IFSTextFileInputStream has been replaced by "IFSFileReader" on page 55.
- com.ibm.as400.access.IFSTextFileOutputStream has been replaced by "IFSFileWriter" on page 57.

Compatibility

The IBM Toolbox for Java no longer ships the x4j400.jar (IBM XML parser). It is recommended that applications use one of the following JAXP-compliant XML parsers:

- The XML parser that is built into JDK 1.4 and higher
- The Apache Xerces XML parser available from xml.apache.org
- One of the XML parsers that ship on i5/OS under /QIBM/ProdData/OS400/xml/lib

IBM Toolbox for Java no longer supports running in the default JVM in Netscape Navigator or Microsoft® Internet Explorer. Running applets that use IBM Toolbox for Java classes in a browser requires that you install a plug-in such as the most recent Sun Java 2 Runtime Environment (JRE) plug-in .

IBM Toolbox for Java no longer includes data400.jar. The classes that were in data400.jar are now in jt400.jar. Remove data400.jar from your CLASSPATH statements.

You cannot use this release of IBM Toolbox for Java to deserialize some objects that you serialized using releases before V5R1.

If you are using Secure Sockets Layer (SSL) to encrypt data flowing between the client and the server, you must use Java Secure Socket Extension (JSSE).

To use all the IBM Toolbox for Java classes, use the Java 2 Platform, Standard Edition (J2SE). Using the vaccess classes or the Graphical Toolbox requires using the Swing package, which comes with J2SE. Using PDML requires that you run version 1.4 or later of the Java Runtime Environment.

For more information, review the i5/OS requirements for running IBM Toolbox for Java.

How to see what's new or changed

To help you see where technical changes have been made, this information uses:

- The » image to mark where new or changed information begins.
- The « image to mark where new or changed information ends.

To find other information about what's new or changed this release, see the Memo to users.

Printable PDF

View or download a PDF of the IBM Toolbox for Java topic. You can also download the IBM Toolbox for Java category in a zipped package.

To view or download the PDF version, select IBM Toolbox for Java PDF (about 6.7 megabytes).

Note: The IBM Toolbox for Java topic contains some information that is not included in the PDF files.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

- Right-click the PDF in your browser (right-click the link above).
- Click **Save Target As...** if you are using Internet Explorer. Click **Save Link As...** if you are using Netscape Communicator.
- Navigate to the directory in which you will save the PDF.
- Click **Save**.

Downloading Adobe Acrobat Reader

You need Adobe Acrobat Reader to view or print these PDFs. You can download a copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

Download IBM Toolbox for Java information in a zipped package

You can download a zipped package of the IBM Toolbox for Java topic that includes the javadocs at the IBM Toolbox for Java and JTOpen Web site .

Note: The information in the zipped package has links to documents that are not included in the zipped package, so these links do not work.

Install and manage IBM Toolbox for Java

Using IBM Toolbox for Java makes it easier to write client Java applets, servlets, and applications that access iSeries resources, data, and programs.

Managing your IBM Toolbox for Java installation

You need to install IBM Toolbox for Java only on client systems that use it, or in a location on your network where your clients can access it. Your clients can be personal computers, dedicated workstations, or iSeries systems. It is important to remember that you can configure an iSeries server or a partition of the server to be a client. In the latter case, you need to install IBM Toolbox for Java on the client partition of the server.

You can use any of the following methods (alone or in combination) to install and manage IBM Toolbox for Java:

- Individual management to install and individually manage IBM Toolbox for Java on each client
- Network management of a single installation by using your network to install and manage a single, shared installation of IBM Toolbox for Java on a server

The following sections briefly explain how each method affects both performance and manageability. How you choose to develop your Java applications and manage your resources determines which of the methods (or which combination of methods) you use.

Individual management

You can choose to individually manage your IBM Toolbox for Java installations on individual clients. The main advantage of installing IBM Toolbox for Java on individual clients is that it reduces the time that a client takes to start an application that uses IBM Toolbox for Java classes.

The main disadvantage is managing those installations individually. Either a user or an application that you create must track and manage which version of IBM Toolbox for Java is installed on each workstation.

Network management of a single installation

You can also use your network to install and manage a single copy of IBM Toolbox for Java on a server that all your clients can access. This kind of network installation provides the following advantages:

- All clients use the same version of IBM Toolbox for Java
- Updating the single installation of IBM Toolbox for Java benefits all clients
- Individual clients have no maintenance issues, other than setting the same initial CLASSPATH

This kind of installation also has the disadvantage of increasing the time that a client takes to start a IBM Toolbox for Java application. You must also enable your client CLASSPATH to point to your server. You can use iSeries NetServer™, which is integrated into i5/OS, or a different method that enables you to access files on iSeries servers, such as iSeries Access for Windows.

Installing IBM Toolbox for Java

How you install IBM Toolbox for Java depends on how you want to manage your installation. Use these topics to install IBM Toolbox for Java.

i5/OS requirements for IBM Toolbox for Java

Ensure that your environment meets the following requirements.

- Required i5/OS options
- Dependencies on other licensed programs
- Compatibility with different levels of i5/OS
- Native optimizations when running on i5/OS JVM
- Requirements for running ToolboxME for iSeries applications

Note: Before you use IBM Toolbox for Java, make sure to address the workstation requirements that pertain to your environment.

Required i5/OS options:

Running IBM Toolbox for Java in a client/server environment requires that you enable the QUSER user profile, start the host servers, and have TCP/IP running.

- The QUSER user profile must be enabled in order to start the host servers.

- Host servers listen for and accept connection requests from clients. i5/OS Host Servers option (licensed product 5722SS1) is included with the base option of i5/OS. For more information, see Host server administration.
- TCP/IP support, which is integrated into i5/OS, allows you to connect your server to a network. For more information, see TCP/IP.

Starting required i5/OS options

From an iSeries command line, start the required i5/OS options by completing the following steps:

1. Ensure that the QUSER profile is enabled.
2. To start the i5/OS host servers, use the CL Start Host Server command. Type **STRHOSTSVR *ALL** and press **ENTER**.
3. To start the TCP/IP distributed data management (DDM) server, use the CL Start TCP/IP Server command. Type **STRTCPSVR SERVER(*DDM)** and press **ENTER**.

Determining if IBM Toolbox for Java is installed on your server:

Many iSeries servers come with the IBM Toolbox for Java licensed product already installed.

To see if IBM Toolbox for Java is already installed, complete the following steps:

1. In iSeries Navigator, select and sign on to the system that you want to use.
2. In the **Function Tree** (the left pane), expand the system, then expand **Configuration and Service**.
3. Expand **Software**, then expand **Installed Products**.
4. In the **Details** pane (the right pane), look in the **Product** column for 5722jc1. If you see this product, the IBM Toolbox for Java licensed program is installed on the selected server.

   Note: You can also find out if IBM Toolbox for Java is installed by using the CL Go to Menu command (**GO MENU(LICPGM)**), Option 11.

If IBM Toolbox for Java is not installed, you can install the IBM Toolbox for Java licensed product.

If a previous version of IBM Toolbox for Java is installed, first delete the currently installed version then install the IBM Toolbox for Java licensed product. To avoid possible problems, consider backing up your currently installed version of IBM Toolbox for Java before you delete it.

Checking the QUSER profile:

The i5/OS Host Servers start under the QUSER user profile, so you first need to ensure that the QUSER profile is enabled.

Check the QUSER profile

To use the command line to check the QUSER profile, complete the following steps:

1. On an iSeries command line, type `DSPUSRPRF USRPRF(QUSER)` and press **Enter**.
2. Make sure that the **Status** is *ENABLED. If the profile status is not *ENABLED, change the QUSER profile.

Changing the QUSER user profile:

If the QUSER profile is not *ENABLED, you must enable it to start the i5/OS Host Servers. Also, the QUSER profile password cannot be *NONE. If this is true, you must reset it.

To use the command line to enable the QUSER profile, complete the following steps:

1. Type `CHGUSRPRF USRPRF(QUSER)` and press **ENTER**.
2. Change the **Status** field to *ENABLED and press **ENTER**.

The QUSER user profile is now ready to start the i5/OS Host Servers.

Dependencies on other licensed programs:

Depending on how you want to use IBM Toolbox for Java, you may need to install other licensed programs.

Spooled file viewer

When you want to use the spooled file viewer functions (SpooledFileViewer class) of IBM Toolbox for Java, ensure that you have installed host option 8 (AFP™ Compatibility Fonts) on your server.

Note: SpooledFileViewer, PrintObjectPageInputStream, and PrintObjectTransformedInputStream classes work only when connecting to V4R4 or later systems.

Secure Sockets Layer

When you want to use Secure Sockets Layer (SSL), ensure that you have installed the following:

- IBM HTTP Server for iSeries licensed program, 5722-DG1
- i5/OS Option 34 (Digital Certificate Manager)
- IBM Cryptographic Access Provider 128-bit for iSeries, 5722-AC3 (only for releases prior to V5R4)

For more information about SSL, see “Secure Sockets Layer and Java Secure Socket Extension” on page 335.

HTTP server for using applets, servlets, or SSL

If you want to use applets, servlets, or SSL on the iSeries server, you must set up an HTTP server and install the class files on the iSeries server. For more information about the IBM HTTP Server, see the IBM HTTP Server for AS/400® Webmaster’s Guide, GC41-5434, at the following URL:

http://www.ibm.com/eserver/iseries/products/http/docs/doc.htm . The Webmaster’s Guide is available in both HTML and PDF formats.

For information about the Digital Certificate Manager and how to create and work with digital certificates using the IBM HTTP Server, see Digital Certificate Management.

Compatibility with different levels of i5/OS:

Because you can use IBM Toolbox for Java both on your server and your client, the compatibility issues affect both running on a server and connecting from a client back to a server.

Running IBM Toolbox for Java on your servers

To install IBM Toolbox for Java (licensed program 5722-JC1 V5R4M0) on an iSeries system, the server must be running one of the following:

- i5/OS Version 5 Release 4
- i5/OS Version 5 Release 3
- i5/OS Version 5 Release 2

You can install only one version of the IBM Toolbox for Java licensed program on the system. To install a different version, first remove the existing IBM Toolbox for Java licensed program.

Using IBM Toolbox for Java to connect from a client back to the server

You can use different versions of IBM Toolbox for Java on a client and on the server to which you are connecting. To use IBM Toolbox for Java to access data and resources on an iSeries system, the **server to which you are connecting** must be running one of the following:

- i5/OS Version 5 Release 4
- i5/OS Version 5 Release 3
- i5/OS Version 5 Release 2

The following table shows the compatibility requirements for installing IBM Toolbox for Java on and connecting back to different versions of i5/OS.

Note: IBM Toolbox for Java does not support forward compatibility. You cannot install IBM Toolbox for Java on or use it to connect to a server that runs a more recent version of i5/OS. For example, if you are using the version of IBM Toolbox for Java that ships with i5/OS V5R2, you cannot install it on or connect to a server that runs i5/OS V5R4.

LPP	Ships with i5/OS	Installs on i5/OS	Connects back to i5/OS
5722-JC1 V5R2M0	V5R2	V4R5 and later	V4R5 and later
5722-JC1 V5R3M0	V5R3	V5R1 and later	V5R1 and later
5722-JC1 V5R4M0	V5R4	V5R2 and later	V5R2 and later

Native optimizations when running on the iSeries JVM:

Native optimizations are a set of functions that make the IBM Toolbox for Java classes work the way a user would expect them to work when running on i5/OS. The optimizations affect operation of IBM Toolbox for Java only when running on the iSeries JVM.

It is very important to understand that your Java programs use native optimizations only when you use the version of IBM Toolbox for Java that matches the version of i5/OS on your server. The optimizations are:

- Signon: When no userid or password is specified in the AS400 object, the userid and password of the current job are used
- Directly calling i5/OS APIs instead of making socket calls to host servers:
  - Record-level database access, data queues and user space when security requirements are met.
  - Program call and command call when security requirements and thread safety requirements are met.

Note: For best performance, set your JDBC driver property to use the native driver when the Java program and database file are on the same iSeries server.

No change to the Java application is needed to get the optimizations. IBM Toolbox for Java automatically enables the optimizations when appropriate.

Native optimization compatibility requirements

The following table shows which versions of IBM Toolbox for Java LPP and i5/OS you must run to use native optimizations. This table documents only compatibility issues that affect native optimizations. For general compatibility issues, see Compatibility with different levels of i5/OS.

Level of i5/OS	IBM Toolbox for Java LPP required to use native optimizations
V5R2	5722-JC1 V5R2M0

Level of i5/OS	IBM Toolbox for Java LPP required to use native optimizations
V5R3	5722-JC1 V5R3M0
V5R4	5722-JC1 V5R4M0

In order to gain the performance improvements, you need to make sure to use the jar file that includes i5/OS native optimizations. For more information, see Note 1 in Jar files.

When the versions of IBM Toolbox for Java and i5/OS do not match, native optimizations are not available. In this case, IBM Toolbox for Java works as if it is running on a client.

ToolboxME for iSeries requirements:

Your workstation, wireless device, and server must meet certain requirements (listed below) for developing and running ToolboxME for iSeries applications. Although IBM Toolbox for Java 2 Micro Edition is considered a part of IBM Toolbox for Java, it is not included in the licensed product.

ToolboxME for iSeries (jt400Micro.jar) is included in the open source version of Toolbox for Java, called JTOpen. You must separately download and set up ToolboxME for iSeries, which is contained in JTOpen.

Requirements

To use ToolboxME for iSeries, your workstation, Tier0 wireless device, and server must meet the following requirements.

Workstation requirements

Workstation requirements for developing ToolboxME for iSeries applications:
- Java 2 Platform, Standard Edition, version 1.3 or higher
- Java virtual machine for wireless devices
- Wireless device simulator or emulator

Wireless device requirements

The only requirement for running ToolboxME for iSeries applications on your Tier0 device is using a Java virtual machine for wireless devices.

Server requirements

Server requirements for using ToolboxME for iSeries applications:
- MEServer class, which is included in IBM Toolbox for Java or the latest version of JTOpen
- i5/OS requirements for IBM Toolbox for Java

Workstation requirements for IBM Toolbox for Java

Ensure that your workstation meets the following requirements.

Note: Before you use IBM Toolbox for Java, make sure to address the i5/OS requirements that pertain to your environment.

Workstation requirements for running IBM Toolbox for Java applications:

To develop and run IBM Toolbox for Java applications, ensure that your workstation meets the following requirements.

- We recommend using a supported Java 2 Standard Edition (J2SE) Java virtual machine. Many new IBM Toolbox for Java functions require using version 1.4 or higher of the JVM.
- Using the vaccess classes or the Graphical Toolbox requires Swing, which comes with J2SE. You can also download Swing 1.1 from the Sun Java Foundation Classes Web site. The following environments have been tested:
  - Windows 2000
  - Windows XP
  - AIX® Version 4.3.3.1
  - Sun Solaris Version 5.7
  - i5/OS Version 5 Release 2 or later
  - Linux® (Red Hat 7.0)
- TCP/IP installed and configured

Workstation requirements for running IBM Toolbox for Java applets:

To develop and run IBM Toolbox for Java applications, ensure that your workstation meets the following requirements.

- A browser that has a compatible Java virtual machine (JVM). The following environments have been tested:
  - Netscape Communicator 4.7, using the Java 1.3 or later plug-in

  Note: IBM Toolbox for Java no longer supports running in the default JVM in Netscape Navigator or Microsoft Internet Explorer. For your applet that uses IBM Toolbox for Java classes to run in a browser, you should install a plug-in such as the Sun Java 2 Runtime Environment (JRE) plug-in .
- TCP/IP installed and configured
- The workstation must connect to a server that is running i5/OS V5R2 or later

Workstation Swing requirements for IBM Toolbox for Java:

IBM Toolbox for Java switched to support Swing 1.1 in V4R5, and this release continues that support. Switching to Swing required programming changes in the IBM Toolbox for Java classes. So, if your programs use the Graphical Toolbox or the vaccess classes from releases before V4R5, you will need to change your programs as well.

In addition to a programming change, the Swing classes must be in the CLASSPATH when the program is run. The Swing classes are part of the Java 2 Platform. If you don't have the Java 2 Platform, you can download the Swing 1.1 classes from Sun Microsystems, Inc.

Installing IBM Toolbox for Java on an iSeries server

You need to install IBM Toolbox for Java on your iSeries server only when you have configured the server or a partition of the server as a client.

Note: The native version of IBM Toolbox for Java ships with i5/OS. So, if you want to use IBM Toolbox for Java only on your iSeries server, you do not have to install the licensed product. For more information about the native version of IBM Toolbox for Java, see Jar files: Note 1.

Before installing IBM Toolbox for Java, you need to ensure that your version of i5/OS meets the requirements for running IBM Toolbox for Java. Also, some servers come preconfigured with an installation of IBM Toolbox for Java. You may want to determine whether the IBM Toolbox for Java licensed product is already installed on your server.

Installing IBM Toolbox for Java

You can install the IBM Toolbox for Java licensed program by using either iSeries Navigator or the command line.

Using iSeries Navigator to install IBM Toolbox for Java

To install IBM Toolbox for Java using iSeries Navigator, complete the following steps:

1. In iSeries Navigator, sign on to the system that you want to use.
2. In the Function Tree (the left pane), expand **My Connections**.
3. Under **My Connections**, right click the system where you want to install IBM Toolbox for Java.
4. Select **Run command**.
5. In the **Restore Licensed Program (RSTLICPGM)** dialog, type the following information, then click **OK**:
   - Product: 5722JC1
   - Device: The name of the device or save file

   Note: For more information, click **Help** in the **Restore Licensed Program (RSTLICPGM)** dialog,

You can use iSeries Navigator to view the status of the resulting Management Central Command task by completing the following steps:

1. Expand **Management Central**.
2. Expand **Task Activity**.
3. Under **Task Activity**, select **Commands**.
4. In the Detail pane, click on the appropriate **Run Command** task.

Using the command line to install IBM Toolbox for Java

To install IBM Toolbox for Java from an iSeries command line, complete the following steps:

1. On an iSeries command line, use the CL Go to Menu command. Type **GO MENU(LICPGM**) and press **ENTER**.
2. Select **11. Install licensed program**.
3. Select **5722-JC1 IBM Toolbox for Java**.

For more information on installing licensed programs, see Managing software and licensed programs.

Installing IBM Toolbox for Java on your workstation

Before you install IBM Toolbox for Java, make sure to address the workstation requirements that pertain to your environment.

How you install IBM Toolbox for Java on your workstation depends on how you want to manage your installation:

- To install IBM Toolbox for Java on individual clients, copy the JAR files to your workstation and configure your workstation CLASSPATH.
- To use IBM Toolbox for Java that is installed on a server, you only have to configure your workstation CLASSPATH to point to the server installation. To point your workstation CLASSPATH to the server, your server must have iSeries Netserver installed.

This documentation explains how to copy the class files to your workstation. For more information about setting the CLASSPATH on your workstation, refer to the operating system documentation for your workstation or information available at the Sun Java Web site  .

Note: Using the IBM Toolbox for Java classes in your application also requires that your system meets the requirements for i5/OS.

The IBM Toolbox for Java class files are packaged in several jar files, consequently you need to copy one or more of these jar files to your workstation. For more information about which jar files are required for specific IBM Toolbox for Java functions, see Jar files.

Example: Copying jt400.jar

The following example assumes you want to copy jt400.jar, which contains the core IBM Toolbox for Java classes.

To manually copy the jar file, complete the following steps:

1. Find the jt400.jar file in the following directory: /QIBM/ProdData/HTTP/Public/jt400/lib
2. Copy jt400.jar from the server to your workstation. You can do this in a variety of ways:
   - Use iSeries Access for Windows to map a network drive on your workstation to the server, then copy the file.
   - Use file transfer protocol (FTP) to send the file (in binary mode) to your workstation.
3. Update the CLASSPATH environment variable of your workstation.
   - For example, if you are using Windows NT® and you copied jt400.jar to C:\jt400\lib, add the following string to the end of your CLASSPATH:

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

You also have the option of using the open source version of IBM Toolbox for Java, called JTOpen. For more information about JTOpen, see the IBM Toolbox for Java and JTOpen Web site .

Jar files:

The IBM Toolbox for Java is shipped as a set of jar files. Each jar file contains Java packages that provide specific functions. You can reduce the amount of required storage space by using only the jar files required to enable the specific functions that you want.

To use a jar file, make sure that you include an entry for it in your CLASSPATH.

The following chart indicates which jar files you must add to your CLASSPATH to use the associated function or package.

IBM Toolbox for Java package or function	Jar files required to be in your CLASSPATH
Access classes	jt400.jar (client) or jt400Native.jar (server) Note 1, or jt400Proxy.jar in a proxy environment
"CommandHelpRetriever class" on page 237	jt400.jar (client) or jt400Native.jar (server) Note 1 and an XML parser and XSLT processor Note 2
CommandPrompter Note 3	jt400.jar, jui400.jar, util400.jar Note 4, and an XML parser Note 2
Commtrace classes	jt400.jar (client) or jt400Native.jar (server) Note 1
HTML classes	jt400.jar Note 1 plus jt400Servlet.jar (client), or jt400Native.jar (server) Note 1
HTMLDocument class	The same jars required for HTML classes, plus an XML parser and XSLT processor Note 2
JCA classes	jt400.jar (client) or jt400Native.jar (server) Note 1
JDBC Data Source GUI	jt400.jar (client) Note 1 and jui400.jarNote 5

IBM Toolbox for Java package or function	Jar files required to be in your CLASSPATH
NLS system and error messages	jt400Mri_lang_cntry.jar Note 6
PCML (development and run-time, parsed) Note 7	jt400.jar (client) or jt400Native.jar (server) Note 1, Note 8, and an XML parser Note 2
PCML (run-time, serialized)	jt400.jar (client) or jt400Native.jar (server) Note 1, Note 8
PDML (development) Note 3	uitools.jar, jui400.jar, util400.jar Note 4, and an XML parser Note 2
PDML (run-time, parsed) Note 3	jui400.jar, util400.jar Note 4, and an XML parser Note 2
PDML (run-time, serialized) Note 3	jui400.jar, and util400.jar Note 4
ReportWriter classes	jt400.jar (client) or jt400Native.jar (server) Note 1, reportwriter jars Note 9, and an XML parser and XSLT processor Note 2
Resource classes	jt400.jar (client) or jt400Native.jar (server) Note 1
RFML	jt400.jar (client) or jt400Native.jar (server) Note 1, and an XML parser Note 2
Security classes	jt400.jar (client) or jt400Native.jar (server) Note 1, or jt400Proxy.jar in a proxy environment
Servlet classes	jt400.jar Note 1 plus jt400Servlet.jar (client), or jt400Native.jar (server) Note 1
iSeries System Debugger Note 3	jt400.jar (client) Note 1 and tes.jar
ToolboxME for iSeries	jt400Micro.jar (client) Note 10 and jt400.jar (server) or jt400Native.jar (server)Note 1
Vaccess classes	jt400.jar (client) Note 1
XPCML	jt400.jar (client) or jt400Native.jar (server) Note 1, and an XML parser and XSLT processor Note 2

Note 1: Some of the IBM Toolbox for Java classes are in more than one jar file:

- **jt400.jar** - Access, commtrace, JCA, JDBC support, MEServer, PCML, resource, RFML, security, utilities, vaccess, and XPCML.
- **jt400.zip** - Use jt400.jar instead of jt400.zip. jt400.zip is shipped to retain compatibility with previous releases of IBM Toolbox for Java.
- **jt400Access.zip** - The same classes that are in jt400.jar minus the vaccess classes. jtAccess400.zip is shipped to retain compatibility with previous releases of IBM Toolbox for Java. Use jt400.jar or jt400Native.jar instead of jt400Access.zip.
- **jt400Native.jar** - Access, HTML, MEServer, PCML, resource, RFML, security, XPCML, and native optimizations. Native optimizations is a set of classes (fewer than 20) that take advantage of iSeries function when running on the iSeries JVM. Because jt400Native.jar contains the native optimizations, when running on the the iSeries JVM, use jt400Native.jar instead of jt400.jar. jt400Native.jar ships with i5/OS and resides in directory /QIBM/ProdData/OS400/jt400/lib.
- **jt400Native11x.jar** - Use jt400Native.jar instead of jt400Native11x.jar. jt400Native11x.jar is shipped to retain compatibility with previous releases of IBM Toolbox for Java.

Note 2: When you must use an XML parser or XSLT processor, make sure that they are JAXP-compliant. For more information, see the following page:

"XML parser and XSLT processor" on page 393

Note 3: Using CommandPrompter, PDML, or the iSeries System Debugger also requires one additional jar file that is not part of IBM Toolbox for Java: jhall.jar. For more information about downloading jhall.jar, see the Sun JavaHelp(TM) Web site .

Note 4: util400.jar contains iSeries-specific classes for formatting input and for using the command line prompter. Using the CommandPrompter class requires util400.jar. Using PDML does not require util400.jar, but it is useful.

Note 5: jui400.jar contains the classes necessary to use the JDBC DataSource GUI interface. jt400.jar (Note 1) contains the classes necessary for all other JDBC functions.

Note 6: jt400Mri_xx_yy.jar contains translated messages, including strings contained in exception messages, dialogs, and output from other normal processing. In jt400Mri_lang_cntry.jar, lang = the ISO Language Code and cntry = the ISO Country or Region Code used to translate the contained text. In some cases, the ISO Country or Region Code is not used. Installing a particular national language version of the IBM Toolbox for Java licensed program on the iSeries installs the appropriate jt400Mri_lang_cntry.jar file. If the language is not supported, the install defaults to the English version, which is included in the IBM Toolbox for Java jar files.

- For example, installing the German language version of licensed program 5722-JC1 installs the German language jar file, jt400Mri_de.jar.

You can add support for other languages by adding more than one of these jar files to the classpath. Java loads the correct string based on the current locale.

Note 7: Serializing your PCML file during development has two benefits:

1. You need to parse the PCML file only during development and not during run-time
2. Users need fewer jar files in their CLASSPATH to run the application

To parse the PCML file during development, you need both the PCML run-time in data.jar or jt400.jar and the PCML parser in x4j400.jar. To run the serialized application , users need only jt400.jar. For more information see Building iSeries program calls with PCML.

Note 8: Use jt400.jar and jt400Native.jar instead of data400.jar. data400.jar contains the PCML runtime classes, which are now also in jt400.jar and jt400Native.jar (Note 1). data400.jar is shipped to retain compatibility with previous releases of IBM Toolbox for Java.

Note 9: Copies of the ReportWriter classes are in more than one jar file:

- composer.jar
- outputwriter.jar
- reportwriters.jar

If your application streams PCL data to an iSeries spooled file, you must make the access classes available by using the appropriate jar file (Note 1). Creating a spooled file to hold PCL data requires the AS400, OutputQueue, PrintParameterList, and SpooledFileOutputStream classes. For more information, see the ReportWriter classes.

Note 10: jt400Micro.jar does not contain the classes needed to run MEServer, which reside in both jt400.jar and jt400Native.jar (Note 1). jt400Micro.jar is available only from the IBM Toolbox for Java and JTOpen Web site .

System properties

You can specify system properties to configure various aspects of the IBM Toolbox for Java.

For example, you can use system properties to define a proxy server or a level of tracing. System properties are useful for convenient runtime configuration without needing to recompile code. System properties work like environment variables in that when you change a system property during runtime, the change is generally not reflected until the next time you run the application.

There are several ways that you can set system properties:

- **Using the java.lang.System.setProperties() method**

  You can set system properties programmatically by using the java.lang.System.setProperties() method.

  For example, the following code sets the `com.ibm.as400.access.AS400.proxyServer` property to `hqoffice`:

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

- **Using the -D option of the java command**

  Many environments allow you to set system properties when running applications from a command line by using the `-D` option of the `java` command.

  For example, the following program runs the application called Inventory with the `com.ibm.as400.access.AS400.proxyServer` property set to `hqoffice`:

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

- **Using a jt400.properties file**

  In some environments, it may be inconvenient to instruct all users to set their own system properties. As an alternative, you can specify IBM Toolbox for Java system properties in a file called jt400.properties that is searched for as if it is part of the com.ibm.as400.access package. In other words, place the jt400.properties file in a com/ibm/as400/access directory pointed to by the classpath.

  For example, set the `com.ibm.as400.access.AS400.proxyServer` property to `hqoffice` by inserting the following line into the jt400.properties file:

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

  The backslash character (\) functions as an escape character in properties files. Specify a literal backslash character by using two backslashes (\\).

  Modify this sample of a jt400.properties file for your environment.

- **Using a Properties class**

  Some browsers do not load properties files without explicitly changing security settings. However, most browsers do allow properties in .class files, so IBM Toolbox for Java system properties can also be specified by a class called com.ibm.as400.access.Properties which extends java.util.Properties.

  For example, to set the `com.ibm.as400.access.AS400.proxyServer` property to `hqoffice`, use the following Java code:

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

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

  Modify and compile this sample of a Properties.java source file for your environment.

If an IBM Toolbox for Java system property is set using more than one of the mechanisms described above, then the precedence is as follows (in order of decreasing precedence):

1. The system property set programmatically using `java.lang.System.setProperties()`
2. The system property set using the `-D` option of the `java` command
3. The system property set using a Properties class

4. The system property set using a jt400.properties file

IBM Toolbox for Java supports the following system properties:
- "Proxy server properties"
- "Trace properties"
- "CommandCall/ProgramCall properties" on page 16
- "FTP properties" on page 16
- "Connection properties" on page 16

Proxy server properties

Proxy server property	Description
com.ibm.as400.access.AS400.proxyServer	Specifies the proxy server host name and port number, using the format: `hostName:portNumber` The port number is optional.
com.ibm.as400.access.SecureAS400.proxyEncryptionMode	Specifies which portion of the proxy data flow is encrypted by using SSL. Valid values are: • 1 = Proxy client to proxy server • 2 = Proxy server to iSeries • 3 = Proxy client to proxy server and proxy server to iSeries
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval	Specifies how often, in seconds, the proxy server looks 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.clientLifetime	Specifies how long, in seconds, a client can be idle before the proxy server removes references to the objects so the JVM can garbage collect them. The proxy server starts a thread to look for clients that are no longer communicating. Use this property to set how long a client can be idle before performing garbage collection on it.

Trace properties

Trace property	Description
com.ibm.as400.access.Trace.category	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.file	Specifies the file to which trace output is written. The default is to write trace output to System.out.
com.ibm.as400.access.ServerTrace.JDBC	Specifies which trace categories to start on the JDBC server job. For information about supported values, see the JDBC server trace property.

CommandCall/ProgramCall properties

CommandCall/ProgramCall property	Description
com.ibm.as400.access.CommandCall.threadSafe	Specifies whether CommandCalls might 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.ProgramCall.threadSafe	Specifies whether ProgramCalls might 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.

FTP properties

FTP property	Description
com.ibm.as400.access.FTP.reuseSocket	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.

Connection properties

Connection property	Description
com.ibm.as400.access.AS400.signonHandler	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.

Example: Properties File

```
#=========================================================#
# IBM Toolbox for Java                                    #
#---------------------------------------------------------#
# Sample properties file                                  #
#                                                         #
# Name this file jt400.properties and store it in a       #
# com/ibm/as400/access directory that is pointed to by    #
# the classpath.                                          #
#=========================================================#

#---------------------------------------------------------#
# Proxy server system properties                          #
#---------------------------------------------------------#
```

```
# This system property specifies the proxy server host name
# and port number, specified in the format: hostName:portNumber
# The port number is optional.
com.ibm.as400.access.AS400.proxyServer=hqoffice

# This system property specifies which portion of the proxy
# data flow is encrypted via SSL.  Valid values are:
#   1 - Proxy client to proxy server
#   2 - Proxy server to AS/400
#   3 - Proxy client to proxy, and proxy server to AS/400
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=1

# This system property specifies how often, in seconds,
# the proxy server will look for idle connections. The
# proxy server starts a thread to look for clients that are
# no longer communicating. Use this property to set how
# often the thread looks for idle connections.
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval=7200

# This system property specifies how long, in seconds, a
# client can be idle before it is cleaned up. The proxy server
# starts a thread to look for clients that are no longer
# communicating. Use this property to set long a client can
# be idle before it is cleaned up.
com.ibm.as400.access.TunnelProxyServer.clientLifetime=2700


#-------------------------------------------------------#
# Trace system properties                               #
#-------------------------------------------------------#

# This system property specifies which trace categories to enable.
# This is a comma-delimited list containing any combination of trace
# categories.  The complete list of trace categories is defined in
# the Trace class.
com.ibm.as400.access.Trace.category=error,warning,information

# This system property specifies the file to which trace output
# is written.  The default is to write trace output to System.out.
com.ibm.as400.access.Trace.file=c:\\temp\\trace.out


#-------------------------------------------------------#
# Command Call system properties                        #
#-------------------------------------------------------#

# This system property specifies whether CommandCalls should
# be assumed to be thread-safe. If true, all CommandCalls are
# assumed to be thread-safe. If false, all CommandCalls are
# assumed to be non-thread-safe. This property is ignored
# for a given CommandCall object if either
# CommandCall.setThreadSafe(true/false) or
# AS400.setMustUseSockets(true) has been performed on the object.
com.ibm.as400.access.CommandCall.threadSafe=true


#-------------------------------------------------------#
# Program Call system properties                        #
#-------------------------------------------------------#

# This system property specifies whether ProgramCalls should
# be assumed to be thread-safe. If true, all ProgramCalls are
# assumed to be thread-safe. If false, all ProgramCalls are
# assumed to be non-thread-safe. This property is ignored
# for a given ProgramCall object if either
# ProgramCall.setThreadSafe(true/false) or
# AS400.setMustUseSockets(true) has been performed on the object.
```

```
com.ibm.as400.access.ProgramCall.threadSafe=true


#-------------------------------------------------------#
# 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


# End
```

Example: System Properties Class Source File

```
//=========================================================
// IBM Toolbox for Java
//---------------------------------------------------------
// Sample properties class source file
//
// Compile this source file and store the class file in
// the classpath.
//=========================================================
package com.ibm.as400.access;

public class Properties
extends java.util.Properties
{
   public Properties ()
   {
      /*-------------------------------------------------------*/
      /* Proxy server system properties                        */
      /*-------------------------------------------------------*/

      // This system property specifies the proxy server host name
      // and port number, specified in the format: hostName:portNumber
      // The port number is optional.
      put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");

      // This system property specifies which portion of the proxy
      // data flow is encrypted via SSL.  Valid values are:
      //   1 - Proxy client to proxy server
      //   2 - Proxy server to iSeries server
      //   3 - Proxy client to proxy, and proxy server to iSeries server
      put("com.ibm.as400.access.SecureAS400.proxyEncryptionMode", "1");

      // This system property specifies how often, in seconds,
      // the proxy server will look for idle connections. The
      // proxy server starts a thread to look for clients that are
      // no longer communicating. Use this property to set how
```

```
// often the thread looks for idle connections.
put("com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval", "7200");

// This system property specifies how long, in seconds, a
// client can be idle before it is cleaned up. The proxy server
// starts a thread to look for clients that are no longer
// communicating. Use this property to set long a client can
// be idle before it is cleaned up.
put("com.ibm.as400.access.TunnelProxyServer.clientLifetime", "2700");

/*-------------------------------------------------------*/
/* Trace system properties                               */
/*-------------------------------------------------------*/

// This system property specifies which trace categories to enable.
// This is a comma-delimited list containing any combination of trace
// categories.  The complete list of trace categories is defined in
// the Trace class.
put ("com.ibm.as400.access.Trace.category", "error,warning,information");

// This system property specifies the file to which trace output
// is written.  The default is to write trace output to System.out.
put ("com.ibm.as400.access.Trace.file", "c:\temp\trace.out");


/*-------------------------------------------------------*/
/* Command Call system properties                        */
/*-------------------------------------------------------*/

// This system property specifies whether CommandCalls should
// be assumed to be thread-safe. If true, all CommandCalls are
// assumed to be thread-safe. If false, all CommandCalls are
// assumed to be non-thread-safe. This property is ignored
// for a given CommandCall object if either
// CommandCall.setThreadSafe(true/false) or
// AS400.setMustUseSockets(true) has been performed on the object.
put ("com.ibm.as400.access.CommandCall.threadSafe", "true");


/*-------------------------------------------------------*/
/* Program Call system properties                        */
/*-------------------------------------------------------*/

// This system property specifies whether ProgramCalls should
// be assumed to be thread-safe. If true, all ProgramCalls are
// assumed to be thread-safe. If false, all ProgramCalls are
// assumed to be non-thread-safe. This property is ignored
// for a given ProgramCall object if either
// ProgramCall.setThreadSafe(true/false) or
// AS400.setMustUseSockets(true) has been performed on the object.
put ("com.ibm.as400.access.ProgramCall.threadSafe", "true");


/*-------------------------------------------------------*/
/* 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");

   }
}
```

IBM Toolbox for Java classes

The IBM Toolbox for Java classes are categorized (like all Java classes) into packages. Each package provides a certain kind of functionality. For convenience, this documentation usually refers to each package with a short name. For example, the com.ibm.as400.access package is called the access package.

Use the links in the following list to find information about the classes in the different IBM Toolbox for Java packages:

- Access classes enable you to access and manage resources on your iSeries
- "Commtrace classes" on page 173 enable you to work with communications trace data for Ethernet or token ring line descriptions
- HTML classes allow you to quickly create HTML forms and tables
- Micro classes enable you to create Java programs that give your wireless devices direct access to iSeries server data and services
- ReportWriter classes allow you to create formatted documents from XML data sources
- Resource classes use a common framework to access and manage iSeries resources
- Security classes make secured connections with the server and verify the identity of a user working on the iSeries server
- Servlet classes assist in retrieving and formatting data for use in Java servlets
- Utility classes enable you to do administrative tasks, such as using the AS400JarMaker class
- Vaccess classes allow you to visually present and manipulate data

Access classes

These classes provide access to resources on your server.

The IBM Toolbox for Java access classes represent iSeries data and resources. The classes work with iSeries servers to provide an internet-enabled interface to access and update server data and resources.

- AS400 - manages sign-on information, creates and maintains socket connections, and sends and receives data
- SecureAS400 - enables you to use an AS400 object when sending or receiving encrypted data
- AS400JPing - allows your Java program to query the host servers to see which services are running and which ports are in service
- "BidiConversionProperties" on page 29 - provides a set of properties that can be used to control the conversion of character set data
- BidiTransform - enables you to do your own conversions of bidirectional text
- "CallStackEntry" on page 29 - represents an entry in the call stack of a specific thread of a server job
- Clustered hash table classes - enables your Java program to share and replicate nonpersistent data among the nodes in highly available clustered hash tables
- Command call - runs iSeries batch commands

- Connection pool - manages a pool of AS400 objects, which is used to share connections and manage the number of connections a user can have to an iSeries server
- Data area - creates, accesses, and deletes data areas
- Data conversion and description - converts and handles data, and describes the record format of a buffer of data
- Data queues - creates, accesses, changes, and deletes data queues
- Digital certificates - manages digital certificates on iSeries servers
- Environment variable - manages iSeries environment variables
- Event log - provides a way to log exceptions and messages independent of the device used to display them
- Exceptions - throws errors when, for example, device errors or programming errors occur
- FTP - provides you with a programmable interface to FTP functions
- Integrated file system - accesses files, opens files, opens input and output streams, and lists the contents of directories
- Java application call - calls a Java program on an iSeries server that runs on the iSeries Java virtual machine
- JDBC - accesses DB2® UDB for iSeries data
- Jobs - accesses iSeries jobs and job logs
- Messages - accesses messages and message queues on the iSeries server
- NetServer configuration - accesses and modifies the state and configuration of the iSeries NetServer
- Permission - displays and changes object authorities on an iSeries server
- Print - manipulates iSeries print resources
- Product license - manage licenses for iSeries products
- Program call - calls an iSeries program
- QSYS object path name - represents objects in the iSeries integrated file system
- Record-level access - creates, reads, updates, and deletes iSeries files and members
- Service program call - calls an iSeries service program
- "SaveFile" on page 41 - represents a save file on a server
- System status - displays system status information and allows access to system pool information
- System values - retrieves and changes system values and network attributes
- "Subsystem" on page 42 - represents a subsystem on the server
- Trace (serviceability) - logs trace points and diagnostic messages
- Users and groups - accesses iSeries users and groups
- User space - accesses an iSeries user space

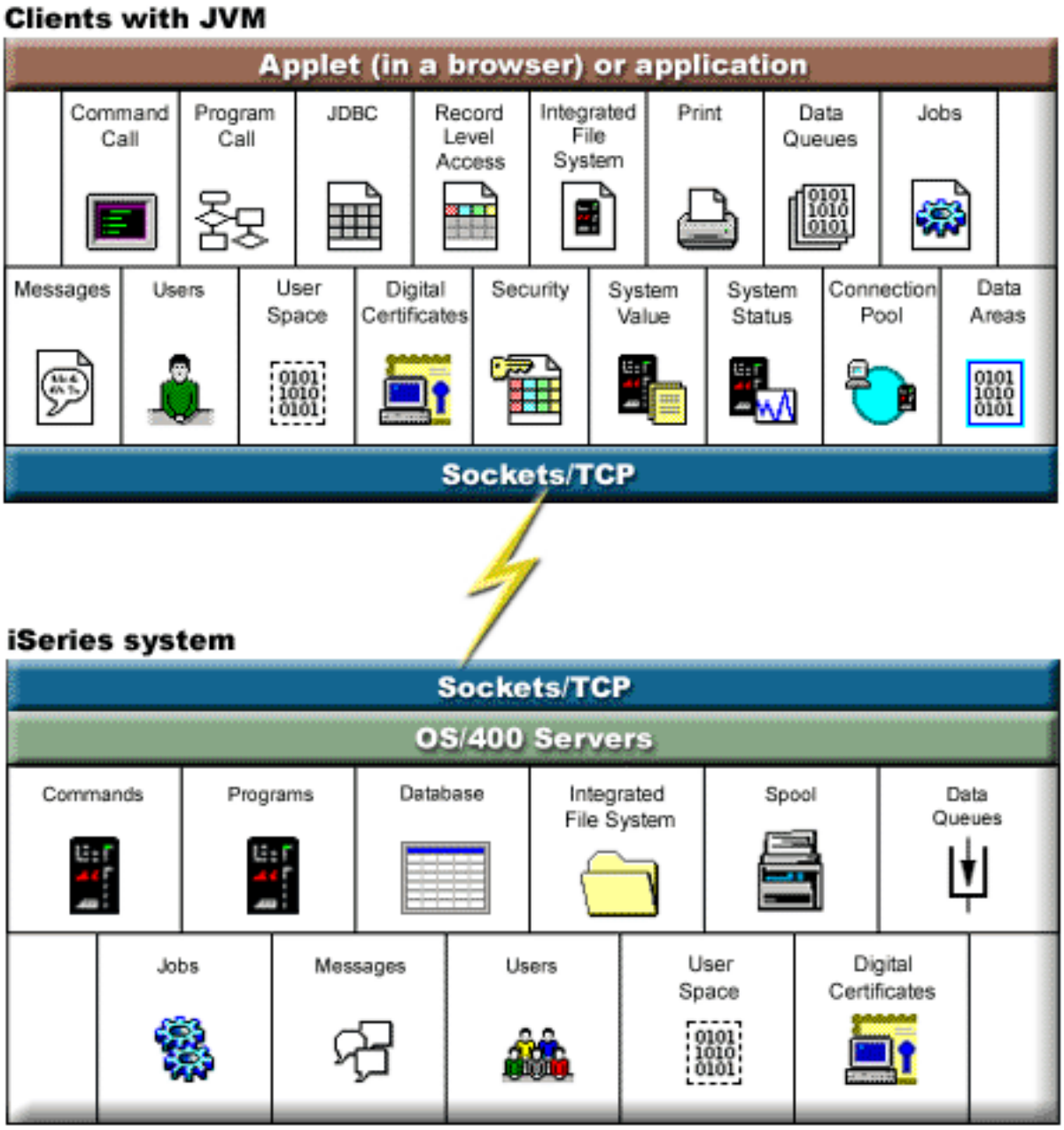
Note: IBM Toolbox for Java provides a second set of classes, called the resource classes, for working with iSeries objects and lists. The resource classes present a generic framework and consistent programming interface for working with various iSeries objects and lists. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application.

Server access points

The IBM Toolbox for Java access classes provide functionality that is similar to using IBM iSeries Access for Windows APIs. However, installing iSeries Access for Windows is not a requirement for using the classes.

The access classes use the existing iSeries servers as the access points. Each server runs in a separate job on the iSeries and sends and receives data streams on a socket connection.

Figure 1: Server access points



Long description of Figure 1: Server access points (rzahh501.gif): found in IBM Toolbox for Java: Server access points

This figure gives a graphic overview of how the classes in the access package of IBM Toolbox for Java use socket connections to interact with data and services on iSeries servers.

Description

The figure is composed of the following:

- A rectangle at the top represents one or more clients, each with a Java virtual machine. The label for the client explains that it is running an applet in a browser or a Java application and has available a socket/TCP connection to the iSeries server.
- A rectangle at the bottom represents an iSeries server. The label for the iSeries server explains that it has one or more servers running i5/OS and those servers have available a socket/TCP connection to the client.
- A lightening bolt connects the two rectangles, and it represents the active socket/TCP connection that allows information to travel between the client and the iSeries servers.

The client (the top rectangle) includes the different functions in the IBM Toolbox for Java access package that you can use to work with data and services on iSeries servers:

- Command call
- Program call
- JDBC
- Record-level access
- Integrated file system
- Print
- Data queues
- Jobs
- Messages
- Users
- User space
- Digital certificates
- Security
- System value
- System status
- Connection pool
- Data areas

The iSeries server (the bottom rectangle) includes the different types of data and services you can work with by using the classes in IBM Toolbox for Java access package:

- Commands
- Programs
- Database
- Integrated file system
- Spool
- Data queues
- Jobs
- Messages
- Users
- User space
- Digital certificates

AS400 class

The AS400 class manages a set of socket connections to the server jobs on server and sign-on behavior for the server, including prompting the user for sign-on information, password caching, and default user management.

The Java program must provide an AS400 object when the Java program uses an instance of a class that accesses the iSeries server. For example, the CommandCall object requires an AS400 object before it can send commands to the iSeries server.

The AS400 object handles connections, user IDs, and passwords differently when it is running in the iSeries Java virtual machine. For more information, see iSeries Java virtual machine.

AS400 objects now support Kerberos authentication, using the Java Generic Security Service Application Programming Interface (JGSS API) to authenticate to the server, instead of using a user ID and password.

Note: Using Kerberos tickets requires that you install J2SDK, v1.4 and configure the Java Generic Security Services (JGSS) Application Programming Interface. For more information about JGSS, see the J2SDK, v1.4 Security Documentation  .

See managing connections for information on managing connections to the iSeries server through the AS400 object. See AS400ConnectionPool for information on reducing initial connect time by requesting connections from a connection pool.

The AS400 class provides the following sign-on functions:

- Authenticate the user profile
- Get a profile token credential and authenticate the associated user profile
- Set a profile token credential
- Manage default user IDs
- Cache passwords
- Prompt for user ID
- Change a password
- Get the version and release of the iSeries

For information about using an AS400 object when sending or receiving encrypted data, see the SecureAS400 class.

Managing default user IDs:

To minimize the number of times a user has to sign on, use a default user ID. The Java program uses the default user ID when a the program does not provide a user ID. The default user ID can be set either by the Java program or through the user interface. If the default user ID has not been established, the Sign-On dialog allows the user to set the default user ID.

Once the default user ID is established for a given server, the Sign-On dialog does not allow the default user ID to be changed. When an AS400 object is constructed, the Java program can supply the user ID and password. When a program supplies the user ID to the AS400 object, the default user ID is not affected. The program must explicitly set the default user ID setUseDefaultUser() if the program wants to set or change the default user ID. See Prompting, default user ID, and password caching summary for more information.

The AS400 object has methods to get, set, and remove the default user ID. The Java program can also disable default user ID processing through the setUseDefaultUser() method. If default user ID processing is disabled and the Java application does not supply a user ID, the AS400 object prompts for user ID every time a connection is made to the iSeries server.

All AS400 objects that represent the same iSeries server within a Java virtual machine use the same default user ID.

In the following example, two connections to the server are created by using two AS400 objects. If the user checked the Default User ID box when signing on, the user is not prompted for a user ID when the second connection is made.

```
                    // Create two AS400 objects to the
                    // same iSeries.
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);
```

The default user ID information is discarded when the last AS400 object for the an iSeries server is garbage collected.

Using a password cache:

The password cache allows the IBM Toolbox for Java to save password and user ID information so that it does not prompt the user for that information every time a connection is made.

Use the methods provided by the AS400 object to do the following:

- Clear the password cache and disable the password cache
- Minimize the number of times a user must type sign-on information

The password cache applies to all AS400 objects that represent an iSeries server within a Java virtual machine. Java does not allow sharing information between virtual machines, so a cached password in one Java virtual machine is not visible to another virtual machine. The cache is discarded when the last AS400 object is garbage collected. The Sign-on dialog has a checkbox that gives the user the option to cache the password. When an AS400 object is constructed, the Java program has the option to supply the user ID and password. Passwords supplied on constructors are not cached.

The AS400 object provides methods to clear the password cache and disable the password cache . See Prompting, default user ID, and password caching summary for more information.

Prompting for user IDs and passwords:

Prompting for user ID and password may occur when connecting to the server. Prompting can be turned off by your Java program.

Java programs can turn off user ID and password prompting and message windows displayed by the AS400 object. An example of when this may be needed is when an application is running on a gateway on behalf of many clients. If prompts and messages are displayed on the gateway machine, the user has no way of interacting with the prompts. These types of applications can turn off all prompting by using the setGuiAvailable() method on the AS400 object.

See Prompting, default user ID, and password caching summary for more information.

Prompting, default user ID, and password caching summary:

Java programs can control when prompting for user ID and password caching occurs. The information from the Sign-On dialog can be used to set the default user ID and cache the password. The following table summarizes when prompting takes place, what information is retrieved, and what information is set.

This table assumes that the Java program allows default user ID processing and password caching, and that you checked the **Default User ID** box and the **Save Password** box on the Sign-On dialog.

Use this table for client connections, not for running Java on your server.

System supplied on constructor	User ID supplied on constructor	Password supplied on constructor	Default user is established	Password in cache for user ID	Result of using marked settings
					User is prompted for system name, user ID, and password. Default user ID is established and password is cached.
Yes					User is prompted for user ID and password. System name is displayed but cannot be changed. Default user ID is established and password is cached.
Yes	**Yes**				User is prompted for password. User ID is displayed and can be changed. System name is displayed but cannot be changed. Default user ID is not changed. Password is cached.
Yes	**Yes**	**Yes**			No prompt. Default user ID is not changed. Password is not cached.

System supplied on constructor	User ID supplied on constructor	Password supplied on constructor	Default user is established	Password in cache for user ID	Result of using marked settings
			Yes		User is prompted for system name and password. User ID is displayed and can be changed. Changing the user ID will not change the default user ID. Password is cached.
Yes			**Yes**		Prompt for password for the default user ID. User ID is displayed and can be changed. System name is displayed but cannot be changed. Password is cached.
Yes			**Yes**	**Yes**	No prompt. Connect using default user ID and password from cache.
Yes	**Yes**			**Yes**	No prompt. Connect as specified user using password from cache.
Yes	**Yes**		**Yes**	**Yes**	No prompt. Connect as specified user using password from cache.
Yes	**Yes**	**Yes**	**Yes**		No prompt. Connect as specified user.

SecureAS400 Class

When an AS400 object communicates with the server, user data (except the user password) is sent unencrypted to the server. So, IBM Toolbox for Java objects associated with an AS400 object exchange data with the server over a normal connection.

When you want to use IBM Toolbox for Java to exchange sensitive data with the server, you can encrypt data by using Secure Sockets Layer (SSL). Use the SecureAS400 object to designate which data you want to encrypt. IBM Toolbox for Java objects associated with a SecureAS400 object exchange data with the server over a secure connection.

For more information, see Secure Sockets Layer and Java Secure Socket Extension.

The SecureAS400 class is a subclass of the AS400 class.

You can set up a secure server connection by creating an instance of a SecureAS400 object in the following ways:

- SecureAS400(String systemName, String userID) prompts you for sign-on information
- SecureAS400(String systemName, String userID, String password) does not prompt you for sign-on information

The following example shows you how to use CommandCall to send commands to the iSeries server using a secure connection:

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

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

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

AS400JPing

The AS400JPing class allows your Java program to query the host servers to see which services are running and which ports are in service.

To query the servers from a command line, use the JPing class.

The AS400JPing class provides several methods:

- Ping the server
- Ping a specific service on the server
- Set a PrintWriter object to which you want to log ping information
- Set the time out for the ping operation

Example: Using AS400JPing

The following example shows how you can use AS400JPing within a Java program to ping the iSeries Remote Command Service:

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

Note: Read the Code example disclaimer for important legal information.

Related information

AS400JPing class

BidiTransform class

This class provides layout transformations that enable you to convert bidirectional text in iSeries format (after first converting it to Unicode) to bidirectional text in Java format, or from Java format to iSeries format.

AS400BidiTransform class

The AS400BidiTransform class allows you to:

- Get and set the system CCSID
- Get and set the string type of iSeries data
- Get and set the string type of Java data
- Convert data from a Java layout to iSeries
- Convert data from an iSeries layout to Java

Example: Using AS400BidiTransform

The following example shows how you can use the AS400BidiTransform class to transform bidirectional text:

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

Note: Read the Code example disclaimer for important legal information.

BidiConversionProperties

The BidiConversionProperties class provides a set of properties that can be used to control the conversion of character set data.

Related information

BidiConversionProperties Javadoc

CallStackEntry

CallStackEntry represents an entry in the call stack of a specific thread of a server job.

Objects of this type are generated by calling `Job.getCallStack()`.

Related information

CallStackEntry Javadoc

ClusteredHashTable classes

The ClusteredHashTable classes enable your Java programs to use highly available clustered hash tables to share and replicate data to nonpersistent storage among the nodes in a cluster.

To use the ClusteredHashTable classes, ensure that you can use nonpersistent storage for the data. Replicated data is not encrypted.

Note: The following information assumes that you understand the concepts and terminology common to iSeries clustering. For more information about clusters and how to use them, see Clusters.

Using the ClusteredHashTable class requires that you have defined and activated a cluster on the iSeries systems. You must also start a clustered has table server. For more information, see Configure clusters and Clustered Hash Table APIs.

Required parameters are the name of the clustered hash table server and the AS400 object, which represents the system that contains the clustered hash table server.

In order to store data in a clustered hash table server, you need a connection handle and a key:

- When you open a connection, the clustered hash table server assigns the connection handle that you must specify on subsequent requests to the clustered hash table server. This connection handle is good only for the instantiated AS400 object, so you must open another connection if you use a different AS400 object.

- You must specify the key to access and change data in the clustered hash table. Duplicate keys are not supported.

The ClusteredHashTable class provides methods that enable you to perform the following actions:
- Open a connection to the clustered hash table server job
- Generate a unique key to store data into the clustered hash table
- Close the active connection to the clustered hash table server job

Some methods in the ClusteredHashTable class use the ClusteredHashTableEntry class to perform the following actions:
- Get an entry from the clustered hash table
- Store an entry in the clustered hash table
- Get a list of entries from the clustered hash table for all user profiles

Example: Using ClusteredHashTable

The following example operates on clustered hash table server named CHTSVR01. It assumes a cluster and a clustered hash table server is already active. It opens a connection, generates a key, puts an entry using the new key in the clustered hash table, gets an entry from the clustered hash table, and closes the connection.

```
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();
```

Using the ClusteredHashTable class causes the AS400 object to connect to the server. For more information, see Managing connections.

Command call

The CommandCall class allows a Java program to call a non-interactive iSeries command.

CommandCall class

Results of the command are available in a list of AS400Message objects.

Input to CommandCall is as follows:

- The command string to run
- The AS400 object that represents the system that will run the command

The command string can be set on the constructor, through the setCommand() method, or on the run() method. After the command is run, the Java program can use the getMessageList() method to retrieve any iSeries messages resulting from the command.

Using the CommandCall class causes the AS400 object to connect to the iSeries. See managing connections for information about managing connections.

When the Java program and the iSeries server command are on the same server, the default IBM Toolbox for Java behavior is to look up the thread safety for the command on the system. If threadsafe, the command is run on-thread. You can suppress the run-time lookup by explicitly specifying thread-safety for the command by using the setThreadSafe() method.

Examples

The following examples show ways you can use the CommandCall class to run different kinds of commands.

Note: Read the Code example disclaimer for important legal information.

Example: Running a command

The following example shows how to use the CommandCall class to run a command on an iSeries server:

```
                    // 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);
```

Example: Running a user-specified command

"Example: Using CommandCall" on page 446 shows how to run a command that is specified by the user.

Connection pooling

Use connection pools to share connections and manage sets (pools) of connections to an iSeries server. For example, an application can retrieve a connection from a pool, use it, then return it to the pool for reuse.

The AS400ConnectionPool class manages a pool of AS400 objects. The AS400JDBCConnectionPool class represents a pool of AS400JDBCConnections that are available for use by a Java program as part of IBM Toolbox for Java support for the JDBC 2.0 Optional Package API. The JDBC ConnectionPool interface is also supported in the JDBC 3.0 API, which is bundled with the Java 2 Platform, Standard Edition, version 1.4.

A connection pool of either type keeps track of the number of connections it creates. Using methods inherited from ConnectionPool, you can set several connection pool properties, including:

- the maximum number of connections that can be given out by a pool
- the maximum lifetime of a connection
- the maximum inactivity time of a connection

In terms of performance, connecting to the server is an expensive operation. Using connection pools can improve performance by avoiding repeated connection times. For example, create connections when you create the connection pool by filling the pool with active (preconnected) connections. Instead of creating new connections, you can use a connection pool to easily retrieve, use, return, and reuse the connection objects.

Retrieve a connection using an AS400ConnectionPool by specifying the system name, user id, the password, and (optionally) the service. To specify the service to which you want to connect, use constants from the AS400 class (FILE, PRINT, COMMAND, and so on).

After retrieving and using the connection, applications return connections to the pool. It is the responsibility of each application to return connections to the pool for reuse. When connections are not returned to the pool, the connection pool continues to grow in size and connections are not reused.

See managing connections for more information about managing when a connection to the iSeries is opened when using the AS400ConnectionPool classes.

Example: Using AS400ConnectionPool

"Example: Using AS400ConnectionPool" on page 447 shows how to reuse AS400 objects.

Data area

The DataArea class is an abstract base class that represents an iSeries data area object

DataArea

This base class has four subclasses that support the following: character data, decimal data, logical data, and local data areas that contain character data.

Using the DataArea class, you can do the following:

- Get the size of the data area
- Get the name of the data area
- Return the AS400 system object for the data area
- Refresh the attributes of the data area
- Set the system where the data area exists

Using the DataArea class causes the AS400 object to connect to the server. See managing connections for information on managing connections.

CharacterDataArea

The CharacterDataArea class represents a data area on the server that contains character data. Character data areas do not have a facility for tagging the data with the proper CCSID; therefore, the data area object assumes that the data is in the user's CCSID. When writing, the data area object converts from a string (Unicode) to the user's CCSID before writing the data to the server. When reading, the data area object assumes that the data is the CCSID of the user and converts from that CCSID to Unicode before returning the string to the program. When reading data from the data area, the amount of data read is by number of characters, not by the number of bytes.

Using the CharacterDataArea class, you can do the following:

- Clear the data area so that it contains all blanks.
- Create a character data area on the system using default property values
- Create a character data area with specific attributes
- Delete the data area from the system where the data area exists
- Return the integrated file system path name of the object represented by the data area.
- Read all of the data that is contained in the data area
- Read a specified amount of data from the data area starting at offset 0 or the offset that you specified
- Set the fully qualified integrated file system path name of the data area
- Write data to the beginning of the data area
- Write a specified amount of data to the data area starting at offset 0 or the offset that you specified

DecimalDataArea

The DecimalDataArea class represents a data area on the server that contains decimal data.

Using the DecimalDataArea class, you can do the following:

- Clear the data area so that it contains 0.0
- Create a decimal data area on the system using default property values
- Create a decimal data area with specified attributes
- Delete the data area from the server where the data area exists
- Return the number of digits to the right of the decimal point in the data area
- Return the integrated file system path name of the object represented by the data area.
- Read all of the data that is contained in the data area
- Set the fully qualified integrated file system path name of the data area
- Write data to the beginning of the data area

Example: Using DecimalDataArea The following example shows how to create and to write to a decimal data area:

Note: Read the Code example disclaimer for important legal information.

```
   // Establish a connection to the server "My400".
AS400 system = new AS400("MyServer");
   // Create a DecimalDataArea object.
QSYSObjectPathName path = new QSYSObjectPathName("MYLIB", "MYDATA", "DTAARA");
DecimalDataArea dataArea = new DecimalDataArea(system, path.getPath());
   // Create the decimal data area on the server using default values.
dataArea.create();
   // Clear the data area.
```

```
dataArea.clear();
   // Write to the data area.
dataArea.write(new BigDecimal("1.2"));
   // Read from the data area.
BigDecimal data = dataArea.read();
   // Delete the data area from the server.
dataArea.delete();
```

LocalDataArea

The LocalDataArea class represents a local data area on the server. A local data area exists as a character data area on the server, but the local data area does have some restrictions of which you should be aware.

The local data area is associated with a server job and cannot be accessed from another job. Therefore, you cannot create or delete the local data area. When the server job ends, the local data area associated with that server job is automatically deleted, and the LocalDataArea object that is referring to the job is no longer valid. You should also note that local data areas are a fixed size of 1024 characters on the server.

Using the LocalDataArea class, you can do the following:

- Clear the data area so that it contains all blanks
- Read all of the data that is contained in the data area
- Read a specified amount of data from the data area starting at offset that you specified
- Write data to the beginning of the data area
- Write a specified amount of data to the data area where the first character is written to offset

LogicalDataArea

The LogicalDataArea class represents a data area on the server that contains logical data.

Using the LogicalDataArea class, you can do the following:

- Clear the data area so that it contains false
- Create a character data area on the server using default property values
- Create a character data area with specified attributes
- Delete the data area from the server where the data area exists
- Return the integrated file system path name of the object represented by the data area.
- Read all of the data that is contained in the data area
- Set the fully qualified integrated file system path name of the data area
- Write data to the beginning of the data area

DataAreaEvent

The DataAreaEvent class represents a data area event.

You can use the DataAreaEvent class with any of the DataArea classes. Using the DataAreaEvent class, you can do the following:

- Get the identifier for the event

DataAreaListener

The DataAreaListener class provides an interface for receiving data area events.

You can use the the DataAreaListener class with any of the DataArea classes. You can invoke the DataAreaListener class when any of the following are performed:

- Clear
- Create
- Delete
- Read
- Write

Data conversion and description

The **data conversion** classes provide the capability to convert numeric and character data between iSeries and Java formats. Conversion may be needed when accessing iSeries data from a Java program. The data conversion classes support conversion of various numeric formats and between various EBCDIC code pages and Unicode.

The **data description** classes build on the data conversion classes to convert all fields in a record with a single method call. The RecordFormat class allows the program to describe data that makes up a DataQueueEntry, ProgramCall parameter, a record in a database file accessed through record-level access classes, or any buffer of iSeries data. The Record class allows the program to convert the contents of the record and access the data by field name or index.

The **converter** classes provide fast and efficient conversion between Java and your iSeries server. BinaryConverter converts between Java byte arrays and Java simple types. CharConverter converts between Java String objects and i5/OS code pages. For more information, see the following:

Converters

Data types

The AS400DataType is an interface that defines the methods required for data conversion. A Java program uses data types when individual pieces of data need to be converted. Conversion classes exist for the following types of data:

- Numeric
- Text (character)
- Composite (numeric and text)

Example: Using AS400DataType classes

The following example illustrates using AS400DataType classes with ProgramCall to supply data for program parameters and to interpret the data returned in program parameters.

Example: Using AS400DataType classes with ProgramCall

Conversion specifying a record format

The IBM Toolbox for Java provides classes for building on the data types classes to handle converting data one record at a time instead of one field at a time. For example, suppose a Java program reads data off a data queue. The data queue object returns a byte array of iSeries data to the Java program. This array can potentially contain many types of iSeries data. The application can convert one field at a time out of the byte array by using the data types classes, or the program can create a record format that describes the fields in the byte array. That record then does the conversion.

Record format conversion can be useful when you are working with data from the program call, data queue, and record-level access classes. The input and output from these classes are byte arrays that can contain many fields of various types. Record format converters can make it easier to convert this data between iSeries format and Java format.

Conversion through record format uses three classes:

- FieldDescription classes identify a field or parameter with a data type and a name.
- A RecordFormat class describes a group of fields.
- A Record class joins the description of a record (in the RecordFormat class) with the actual data.
- A LineDataRecordWriter class writes a record to an OutputStream in line data format

Examples: Using record format conversion classes

The following examples illustrate using the record format conversion classes with data queues:

Using the Record and RecordFormat classes to put data on a queue

Using the FieldDescription, RecordFormat, and Record classes

Conversion classes for numeric data:

Conversion classes for numeric data convert numeric data from the format used on the iSeries server (called **server format** in the following table) to the Java format.

Supported types are shown in the following table:

Numeric Type	Description
AS400Bin2	Converts between a signed two-byte number in the server format to a Java Short object.
AS400Bin4	Converts between a signed four-byte number in the server format and a Java Integer object.
AS400ByteArray	Converts between two byte arrays. This is useful because the converter correctly zero-fills and pads the target buffer.
AS400Float4	Converts between a signed four-byte floating point number in the server format and a Java Float object.
AS400Float8	Converts between a signed eight-byte floating point number in the server format and a Java Double object.
AS400PackedDecimal	Converts between a packed-decimal number in the server format and a Java BigDecimal object.
AS400UnsignedBin2	Converts between an unsigned two-byte number in the server format and a Java Integer object.
AS400UnsignedBin4	Converts between an unsigned four-byte number in the server format and a Java Long object.
AS400ZonedDecimal	Converts between a zoned-decimal number in the server format and a Java BigDecimal object.

Examples

The following examples show data conversions that use a numeric type in the server format and a Java int:

Example: Converting from the server format to a Java int

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

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

// Convert from server type to Java object. The number starts at the
// beginning of the buffer.
Integer intObject = (Integer) bin4Converter.toObject(data,0);

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

Example: Converting from a Java int to the server format

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

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

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

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

Text conversion:

Character data is converted through the AS400Text class. This class converts character data between an EBCDIC code page and character set (CCSID), and Unicode.

AS400Text

When the AS400Text object is constructed, the Java program specifies the length of the string to be converted and the server CCSID or encoding. The CCSID of the Java program is assumed to be 13488 Unicode. The toBytes() method converts from Java form to byte array in iSeries format. The toObject() method converts from a byte array in iSeries format to Java format.

The AS400BidiTransform class provides layout transformations that allow the conversion of bidirectional text in iSeries format (after its conversion to Unicode) to bidirectional text in Java format, or from Java format to iSeries format. The default conversion is based on the CCSID of the job. To alter the direction and shaping of the text, specify a BidiStringType. Note that where IBM Toolbox for Java objects perform the conversion internally, as in the DataArea class, the objects have a method to override the string type. For example, the DataArea class has addVetoableChangeListener() method that you can specify to listen for a veto changes to certain properties, including string type.

Example: Converting text data

The following example assumes that a DataQueueEntry object returns text in EBCDIC. The example converts the EBCDIC data to Unicode, so that the Java program can use data:

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

// Create a converter for the iSeries data type. Note a default
```

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

// Note: Optionally, you can create a converter for a specific
// CCSID. Use an AS400 object in case the program is running
// as 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 iSeries code page matches
// the CCSID returned by the AS400 object.
AS400Text textConverter = new AS400Text(textLength, system);

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

Conversion classes for composite types:

Conversion classes for composite types are as follows.

- AS400Array - Allows the Java program to work with an array of data types.
- AS400Structure - Allows the Java program to work with a structure whose elements are data types.

Example: Converting composite data types

The following example shows conversion from a Java structure to a byte array and back again. The example assumes that the same data format is used for both sending and receiving data.

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

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

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

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

```
// ... send the byte array to the server. Get data back from the
// server.  The returned data will also be a byte array.


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

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

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

FieldDescription classes:

The field description classes allow the Java program to describe the contents of a field or parameter with a data type and a string containing the name of the field. If the program is working with data from record-level access, it can also specify any iSeries data definition specification (DDS) keywords that further describe the field.

Field description classes

The field description classes are as follows:

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

Example: Creating field descriptions

The following example assumes that the entries on a data queue have the same format. Each entry has a message number (AS400Bin4), a time stamp (8 characters), and message text (50 characters) that you can describe with field descriptions:

```
// 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");
```

You can now group the field descriptions in an instance of the RecordFormat class. To see how to add the field descriptions to a RecordFormat object, see the example on the following page:

"RecordFormat class"

RecordFormat class:

The RecordFormat class allows the Java program to describe a group of fields or parameters. A record object contains data described by a RecordFormat object. If the program is using record-level access classes, the RecordFormat class also allows the program to specify descriptions for key fields.

RecordFormat

A RecordFormat object contains a set of field descriptions. The field description can be accessed by index or by name. Methods exist on the RecordFormat class to do the following:

- Add field descriptions to the record format.
- Add key field descriptions to the record format.
- Retrieve field descriptions from the record format by index or by name.
- Retrieve key field descriptions from the record format by index or by name.
- Retrieve the names of the fields that make up the record format.
- Retrieve the names of the key fields that make up the record format.
- Retrieve the number of fields in the record format.
- Retrieve the number of key fields in the record format.
- Create a Record object based on this record format.

Example: Adding field descriptions to a record format

The following example adds the field descriptions created in the field description example to a record format:

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

To see how to create a record from the record format, see the example on the following page:

"Record class"

Record class:

The record class allows the Java program to process data described by the record format class.

Record class

Data is converted between byte arrays containing the server data and Java objects. Methods are provided in the record class to do the following:

- Retrieve the contents of a field, by index or by name, as a Java object.
- Retrieve the number of fields in the record.
- Set the contents of a field, by index or by name, with a Java object.
- Retrieve the contents of the record as server data into a byte array or output stream.
- Set the contents of the record from a byte array or an input stream.
- Convert the contents of the record to a String.

Example: Reading a record

The following example uses the record format created in the record format example:

```
// 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");
```

Retrieving the contents of a field:

Retrieve the contents of a Record object by having your Java program either get one field at a time or get all the fields at once.

Use the getField() method to retrieve a single field by name or by index. Use the getFields() method to retrieve all of the fields as an Object[].

The Java program must cast the Object (or element of the Object[]) returned to the appropriate Java object for the retrieved field. The following table shows the appropriate Java object to cast based on the field type.

Field Type (DDS)	Field Type (FieldDescription)	Java Object
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), single precision	FloatFieldDescription	Float
FLOAT (F), double precision	FloatFieldDescription	Double
HEXADECIMAL (H)	HexFieldDescription	byte[]
PACKED DECIMAL (P)	PackedDecimalFieldDescription	BigDecimal
TIME (T)	TimeDecimalFieldDescription	String
TIMESTAMP (Z)	TimestampDecimalFieldDescription	String
ZONED DECIMAL (P)	ZonedDecimalFieldDescription	BigDecimal

SaveFile:

The SaveFile class represents a save file on a server.

Related information

SaveFile Javadoc

Subsystem:

The Subsystem class represents a subsystem on the server.

Related information

Subsystem Javadoc

Setting the contents of a field:

Set the contents of a Record object by using the setField() method in your Java program.

SetField() method

The Java program must specify the appropriate Java object for the field being set. The following table shows the appropriate Java object for each possible field type.

Field Type (DDS)	Field Type (FieldDescription)	Java Object
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), single precision	FloatFieldDescription	Float
FLOAT (F), double precision	FloatFieldDescription	Double
HEXADECIMAL (H)	HexFieldDescription	byte[]
PACKED DECIMAL (P)	PackedDecimalFieldDescription	BigDecimal
TIME (T)	TimeDecimalFieldDescription	String
TIMESTAMP (Z)	TimestampDecimalFieldDescription	String
ZONED DECIMAL (P)	ZonedDecimalFieldDescription	BigDecimal

LineDataRecordWriter class:

The LineDataRecordWriter class writes the record data, in line data format, to an OutputStream. The class translates the data into bytes by using the specified CCSID. The record format associated with the record determines the format of the data.

LineDataRecordWriter

Using LineDataRecordWriter requires that the following record format attributes be set:

- Record format ID
- Record format type

In conjunction with the Record or the RecordFormat classes, the LineDataRecordWriter takes a record as input to the writeRecord() method. (Record takes a RecordFormat as input when you instantiate it.)

The LineDataRecordWriter class provides methods that allow you to:

- Get the CCSID
- Get the name of the encoding
- Write the record data, in line data format, to an OutputStream

Example: Using the LineDataRecordWriter class

Note: Read the Code example disclaimer for important legal information.

The following example shows one way to use the LineDataRecordWriter class to write a record:

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

Data queues

The DataQueue classes allow the Java program to interact with server data queues.

Data queues on iSeries servers have the following characteristics:

- The data queue allows for fast communications between jobs. Therefore, it is an excellent way to synchronize and pass data between jobs.
- Many jobs can simultaneously access the data queues.
- Messages on a data queue are free format. Fields are not required as they are in database files.

- The data queue can be used for either synchronous or asynchronous processing.
- The messages on a data queue can be ordered in one the following ways:
  - Last-in first-out (LIFO). The last (newest) message that is placed on the data queue is the first message that is taken off the queue.
  - First-in first-out (FIFO). The first (oldest) message that is placed on the data queue is the first message that is taken off the queue.
  - Keyed. Each message on the data queue has a key associated with it. A message can be taken off the queue only by specifying the key that is associated with it.

The data queue classes provide a complete set of interfaces for accessing server data queues from your Java program. It is an excellent way to communicate between Java programs and programs on the server that are written in any programming language.

A required parameter of each data queue object is the AS400 object that represents the server that has the data queue or where the data queue is to be created.

Using the data queue classes causes the AS400 object to connect to the server. See managing connections for information about managing connections.

Each data queue object requires the integrated file system path name of the data queue. The type for the data queue is DTAQ. See integrated file system path names for more information.

Sequential and keyed data queues

The data queue classes support the following data queues:

- Sequential data queues
- Keyed data queues

Methods common to both types of queues are in the BaseDataQueue class. The DataQueue class extends the BaseDataQueue class in order to complete the implementation of sequential data queues. The BaseDataQueue class is extended by the KeyedDataQueue class to complete the implementation of keyed data queues.

When data is read from a data queue, the data is placed in a DataQueueEntry object. This object holds the data for both keyed and sequential data queues. Additional data available when reading from a keyed data queue is placed in a KeyedDataQueueEntry object that extends the DataQueueEntry class.

The data queue classes do not alter data that is written to or is read from the server data queue. The Java program must correctly format the data. The data conversion classes provide methods for converting data.

Example: Using DataQueue and DataQueueEntry

The following example creates a DataQueue object, reads data from the DataQueueEntry object, and then disconnects from the system.

Note: Read the Code example disclaimer for important legal information.

```
// 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);
```

Sequential data queues:

Entries on a sequential data queue on the server are removed in first-in first-out (FIFO) or last-in first-out (LIFO) sequence.

The BaseDataQueue and DataQueue classes provide the following methods for working with sequential data queues:

- Create a data queue on the server. The Java program must specify the maximum size of an entry on the data queue. The Java program can optionally specify additional data queue parameters (FIFO vs LIFO, save sender information, specify authority information, force to disk, and provide a queue description) when the queue is created.
- Peek at an entry on the data queue without removing it from the queue. The Java program can wait or return immediately if no entry is currently on the queue.
- Read an entry off the queue. The Java program can wait or return immediately if no entry is available on the queue.
- Write an entry to the queue.
- Clear all entries from the queue.
- Delete the queue.

The BaseDataQueue class provides additional methods for retrieving the attributes of the data queue.

Examples: Working with sequential data queues

The following sequential data queue examples illustrate how a producer puts items on a data queue and how a consumer takes the items off the queue and processes them:

"Example: Using DataQueue classes to put data on a queue" on page 452

"Example: Using DataQueue classes to read entries off a data queue" on page 455

Keyed data queues:

The BaseDataQueue and KeyedDataQueue classes provide methods for working with keyed data queues.

BaseDataQueue

KeyedDataQueue

- Create a keyed data queue on the server. The Java program must specify key length and maximum size of an entry on the queue. The Java program can optionally specify authority information, save sender information, force to disk, and provide a queue description.
- Peek at an entry based on the specified key without removing it from the queue. The Java program can wait or return immediately if no entry is currently on the queue that matches the key criteria.
- Read an entry off the queue based on the specified key. The Java program can wait or return immediately if no entry is available on the queue that matches the key criteria.
- Write a keyed entry to the queue.
- Clear all entries or all entries that match a specified key.

- Delete the queue.

The BaseDataQueue and KeyedDataQueue classes also provide additional methods for retrieving the attributes of the data queue.

Examples: Working with keyed data queues

The following keyed data queue examples illustrate how a producer puts items on a data queue, and how a consumer takes the items off the queue and processes them:

"Example: Using KeyedDataQueue" on page 460

"Example: Using KeyedDataQueue classes to read entries off a data queue" on page 464

Digital certificates

Digital certificates are digitally-signed statements used for secured transactions over the internet.

Digital certificates can be used on servers running i5/OS Version 4 Release 3 (V4R3) and later. To make a secure connection using the Secure Sockets Layer (SSL), a digital certificate is required.

Digital certificates comprise the following:

- The public encryption key of the user
- The name and address of the user
- The digital signature of a third-party certification authority (CA). The authority's signature means that the user is a trusted entity.
- The issue date of the certificate
- The expiration date of the certificate

As an administrator of a secured server, you can add a certification authority's "trusted root key" to the server. This means that your server will trust anyone who is certified through that particular certification authority.

Digital certificates also offer encryption, ensuring a secure transfer of data through a private encryption key.

You can create digital certificates through the javakey tool. (For more information about javakey and Java security, see the Sun Microsystems, Inc., Java Security page  .) The IBM Toolbox for Java licensed program has classes that administer digital certificates on the iSeries server.

The AS400Certificate classes provide methods to manage X.509 ASN.1 encoded certificates. Classes are provided to do the following:

- Get and set certificate data.
- List certificates by validation list or user profile.
- Manage certificates, for example, add a certificate to a user profile or delete a certificate from a validation list.

Using a certificate class causes the AS400 object to connect to the server. See managing connections for information about managing connections.

On the server, certificates belong to a validation list or to a user profile.

- The AS400CertificateUserProfileUtil class has methods for managing certificates on a user profile.
- The AS400CertificateVldlUtil class has methods for managing certificates in a validation list.

Using AS400CertificateUserProfileUtil and AS400CertificateVldlUtil requires that you install base operating system option 34 (Digital Certificate Manager). These two classes extend AS400CertificateUtil, which is an abstract base classes that defines methods common to both subclasses.

The AS400Certificate class provides methods to read and write certificate data. Data is accessed as an array of bytes. The Java.Security package in Java virtual machine 1.2 provides classes that can be used to get and set individual fields of the certificate.

Listing certificates

To get a list of certificates, the Java program must do the following:

1. Create an AS400 object.
2. Construct the correct certificate object. Different objects are used for listing certificates on a user profile (AS400CertificateUserProfileUtil) versus listing certificates in a validation list (AS400CertificateVldlUtil).
3. Create selection criteria based on certificate attributes. The AS400CertificateAttribute class contains attributes used as selection criteria. One or more attribute objects define the criteria that must be met before a certificate is added to the list. For example, a list might contain only certificates for a certain user or organization.
4. Create a user space on the server and put the certificate into the user space. Large amounts of data can be generated by a list operation. The data is put into a user space before it can be retrieved by the Java program. Use the listCertificates() method to put the certificates into the user space.
5. Use the getCertificates() method to retrieve certificates from the user space.

Example: Listing digital certificates

The following example lists certificates in a validation list. It lists only those certificates belonging to a certain person.

Note: Read the Code example disclaimer for important legal information.

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

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

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

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

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

   // Process the certificates
```

EnvironmentVariable class

The EnvironmentVariable class and the EnvironmentVariableList class enable you to access and set iSeries system-level environment variables.

EnvironmentVariable class

EnvironmentVariableList class

Each variable has unique identifiers: the system name and the environment variable name. Each environment variable is associated with a CCSID, which is by default the CCSID of the current job, which describes where the contents of the variable are stored.

Note: Environment variables are different than system values, although they are often used for the same purpose. See SystemValues for more information on how to access system values.

Use an EnvironmentVariable object to perform the following actions on an environment variable:

- Get and set the name
- Get and set the system
- Get and set the value (which allows you to change the CCSID)
- Refresh the value

Example: Creating, setting, and getting environment variables

The following example creates two EnvironmentVariables and sets and gets their values.

Note: Read the Code example disclaimer for important legal information.

```
// Create the iSeries system object.
AS400 system = new AS400("mySystem");

// Create the foreground color environment variable and set it to red.
EnvironmentVariable fg = new EnvironmentVariable(system, "FOREGROUND");
fg.setValue("RED");

// Create the background color environment variable and get its value.
EnvironmentVariable bg = new EnvironmentVariable(system, "BACKGROUND");
String background = bg.getValue();
```

Exceptions

The IBM Toolbox for Java access classes throw exceptions when device errors, physical limitations, programming errors, or user input errors occur. The exception classes are based upon the type of error that occurs instead of the location where the error originates.

Most exceptions contain the following information:

- **Error type:** The exception object that is thrown indicates the type of error that occurred. Errors of the same type are grouped together in an exception class.
- **Error details:** The exception contains a return code to further identify the cause of the error that occurred. The return code values are constants within the exception class.
- **Error text:** The exception contains a text string that describes the error that occurred. The string is translated in the locale of the client Java virtual machine.

Example: Catching a thrown exception

The following example shows how to catch a thrown exception, retrieve the return code, and display the exception text:

Note: Read the Code example disclaimer for important legal information.

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

FTP class

The FTP class provides a programmable interface to FTP functions.

FTP class

You are no longer required to use java.runtime.exec() or tell your users to run FTP commands in a separate application. That is, you can program FTP functions directly into your application. So, from within your program, you can do the following:

- Connect to an FTP server
- Send commands to the server
- List the files in a directory
- Get files from the server **and**
- Put files to the server

Example: Using FTP to copy files from a server

Note: Read the Code example disclaimer for important legal information.

For example, with the FTP class, you can copy a set of files from a directory on a server:

```
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 is a generic interface that works with many different FTP servers. Therefore, it is up to the programmer to match the semantics of the server.

AS400FTP subclass

While the FTP class is a generic FTP interface, the AS400FTP subclass is written specifically for the FTP server on the server. That is, it understands the semantics of the FTP server on the iSeries server. For example, this class understands the various steps needed to transfer a save file to the server and performs these steps automatically. AS400FTP also ties into the security facilities of the IBM Toolbox for Java. As with other IBM Toolbox for Java classes, AS400FTP depends on the AS400 object for system name, user ID, and password.

Example: Using AS400FTP to save a file to the server

Note: Read the Code example disclaimer for important legal information.

The following example puts a save file to the server. Note the application does not set data transfer type to binary or use CommandCall to create the save file. Since the extension is .savf, AS400FTP class detects the file to put is a save file so it does these steps automatically.

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

Integrated file system

The integrated file system classes allow a Java program to access files in the integrated file system of an iSeries server as a stream of bytes or a stream of characters. The integrated file system classes were created because the java.io package does not provide file redirection and other iSeries functionality.

The function that is provided by the IFSFile classes is a superset of the function provided by the file IO classes in the java.io package. All methods in java.io FileInputStream, FileOutputStream, and RandomAccessFile are in the integrated file system classes.

In addition, the classes contain methods to do the following:

- Specify a file sharing mode to deny access to the file while it is in use
- Specify a file creation mode to open, create, or replace the file
- Lock a section of the file and deny access to that part of the file while it is in use
- List the contents of a directory more efficiently
- Cache the contents of a directory to improve performance by limiting calls to the server
- Determine the number of bytes available on the server file system
- Allow a Java applet to access files in the server file system
- Read and write data as text instead of as binary data
- Determine the type of the file object (logical, physical, save, and so on) when the object is in the QSYS.LIB file system

Through the integrated file system classes, the Java program can directly access stream files on the iSeries. The Java program can still use the java.io package, but the client operating system must then provide a method of redirection. For example, if the Java program is running on a Windows 95 or Windows NT operating system, the Network Drives function of iSeries Access for Windows is required to redirect java.io calls to the iSeries. With the integrated file system classes, you do not need iSeries Access for Windows.

A required parameter of the integrated file system classes is the AS400 object that represents the iSeries system that contains the file. Using the integrated file system classes causes the AS400 object to connect to the iSeries. See managing connections for information about managing connections.

The integrated file system classes require the hierarchical name of the object in the integrated file system. Use the forward slash as the path separator character. The following example shows how to access FILE1 in directory path DIR1/DIR2:

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

Integrated file system classes

The following table lists the integrated file system classes:

Integrated file system class	Description
IFSFile	Represents a file in the integrated file system
IFSJavaFile	Represents a file in the integrated file system (extends java.io.File)
IFSFileInputStream	Represents an input stream for reading data from an iSeries file
IFSTextFileInputStream	Represents a stream of character data read from a file (deprecated)
"IFSFileReader" on page 55	Use this class for reading character files in the integrated file system.
IFSFileOutputStream	Represents an output stream for writing data to an iSeries file
"IFSFileWriter" on page 57	Use IFSFileWriter for writing character files in the integrated file system.
IFSTextFileOutputStream	Represents a stream of character data being written to a file (deprecated)
IFSRandomAccessFile	Represents a file on the iSeries for reading and writing data
IFSFileDialog	Allows the user to move within the file system and to select a file within the file system
"IFSSystemView" on page 60	IFSSystemView provides a gateway to the iSeries integrated file system, for use when constructing javax.swing.JFileChooser objects.

Examples: Using integrated file system classes

"Example: Using IFS classes to copy a file from one directory to another" on page 471 shows how to use the integrated file system classes to copy a file from one directory to another on the iSeries.

"Example: Using the IFS classes to list the contents of a directory" on page 474 shows how to use the integrated file system classes to list the contents of a directory on the iSeries.

IFSFile class:

The IFSFile class represents an object in the iSeries integrated file system.

IFSFile

The methods on IFSFile represent operations that are done on the object as a whole. You can use IFSFileInputStream, IFSFileOutputStream, and IFSRandomAccessFile to read and write to the file. The IFSFile class allows the Java program to do the following:

- Determine if the object exists and is a directory or a file
- Determine if the Java program can read from or write to a file
- Determine the length of a file
- Determine the permissions of an object and set the permissions of an object
- Create a directory
- Delete a file or directory
- Rename a file or directory
- Get or set the last modification date of a file
- List the contents of a directory
- List the contents of a directory and save the attribute information to a local cache
- Determine the amount of space available on the system
- Determine the type of the file object when it is in the QSYS.LIB file system

You can get the list of files in a directory by using either the list() method or the listFiles() method:

- The listFiles() method caches information for each file on the initial call. After calling listFiles(), using other methods to query file details results in better performance because the information is retrieved from the cache. For example, calling isDirectory() on an IFSFile object returned from listFiles() does not require a call to the server.
- The list() method retrieves information about each file in a separate request to the server, making it slower and more demanding of server resources.

Note: Using listFiles() means that the information in the cache may eventually become stale, so you may need to refresh the data by calling listFiles() again.

Examples

The following examples show how to use the IFSFile class:

- "Example: Creating a directory" on page 467
- "Example: Using IFSFile exceptions to track errors" on page 467
- "Example: Listing files with a .txt extension" on page 468
- "Example: Using the IFSFile listFiles() method to list the contents of a directory" on page 469

IFSJavaFile class:

This class represents a file in the iSeries integrated file system and extends the java.io.File class. IFSJavaFile allows you to write files for the java.io.File interface that access iSeries integrated file systems.

IFSJavaFile

IFSJavaFile makes portable interfaces that are compatible with java.io.File and uses only the errors and exceptions that java.io.File uses. IFSJavaFile uses the security manager features from java.io.File, but unlike java.io.File, IFSJavaFile uses security features continuously.

You use IFSJavaFile with IFSFileInputStream and IFSFileOutputStream. It does not support java.io.FileInputStream and java.io.FileOutputStream.

IFSJavaFile is based on IFSFile; however, its interface is more like java.io.File than IFSFile. IFSFile is an alternative to the IFSJavaFile class.

You can get the list of files in a directory by using either the list() method or the listFiles() method:

- The listFiles() method performs better because it retrieves and caches information for each file on the initial call. After that, information on each file is retrieved from the cache.
- The list() method retrieves information on each file in a separate request, making it slower and more demanding of server resources.

Note: Using listFiles() means that the information in the cache eventually become stale, so you may need to refresh the data.

Example: Using IFSJavaFile

Note: Read the Code example disclaimer for important legal information.

The following example shows how to use the IFSJavaFile class:

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

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

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

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

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

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

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

IFSFileInputStream:

The IFSFileInputStream class represents an input stream for reading data from a file on the server.

IFSFileInputStream

As in the IFSFile class, methods exist in IFSFileInputStream that duplicate the methods in FileInputStream from the java.io package. In addition to these methods, IFSFileInputStream has additional methods specific to iSeries servers. The IFSFileInputStream class allows a Java program to do the following:

- Open a file for reading. The file must exist since this class does not create files on the server. You can use a constructor that allows you to specify the file sharing mode.
- Determine the number of bytes in the stream.
- Read bytes from the stream.
- Skip bytes in the stream.
- Lock or unlock bytes in the stream.
- Close the file.

As in FileInputStream in java.io, this class allows a Java program to read a stream of bytes from the file. The Java program reads the bytes sequentially with only the additional option of skipping bytes in the stream.

In addition to the methods in FileInputStream, IFSFileInputStream gives the Java program the following options:

- Locking and unlocking bytes in the stream. See IFSKey for more information.
- Specifying a sharing mode when the file is opened. See sharing modes for more information.

Example: Using IFSFileInputStream

The following example shows how to use the IFSFileInputStream class.

```
                    // 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();
```

IFSTextFileInputStream class:

IFSTextFileInputStream has been deprecated and replaced by class IFSFileReader.

The IFSTextFileInputStream class represents a stream of character data read from a file. The data read from the IFSTextFileInputStream object is supplied to the Java program in a Java String object, so it is always Unicode. When the file is opened, the IFSTextFileInputStream object determines the CCSID of the

data in the file. If the data is stored in an encoding other than Unicode, the IFSTextFileInputStream object converts the data from the file's encoding to Unicode before giving the data to the Java program. If the data cannot be converted, an UnsupportedEncodingException is thrown.

The following example shows how to use the IFSTextFileInputStream:

```
                    // 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();
```

Related reference

"IFSFileReader"
Use this class for reading character files in the integrated file system.

IFSFileReader:

Use this class for reading character files in the integrated file system.

IFSFileReader is meant for reading streams of characters. IFSFileReader replaces IFSTextFileOutputStream.

Example: Using IFSFileReader

The following example illustrates the use of IFSFileReader:

```
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();
```

Related information

IFSFileReader Javadoc

IFSFileOutputStream class:

The IFSFileOutputStream class represents an output stream for writing data to a file on the server.

IFSFileOutputStream

As in the IFSFile class, methods exist in IFSFileOutputStream that duplicate the methods in FileOutputStream from the java.io package. IFSFileOutputStream also has additional methods specific to the server. The IFSFileOutputStream class allows a Java program to do the following:

- Open a file for writing. If the file already exists, it is replaced. Constructors are available that specify the file sharing mode and whether the contents of an existing file have been appended.
- Write bytes to the stream.
- Commit to disk the bytes that are written to the stream.
- Lock or unlock bytes in the stream.
- Close the file.

As in FileOutputStream in java.io, this class allows a Java program to sequentially write a stream of bytes to the file.

In addition to the methods in FileOutputStream, IFSFileOutputStream gives the Java program the following options:

- Locking and unlocking bytes in the stream. See IFSKey for more information.
- Specifying a sharing mode when the file is opened. See sharing modes for more information.

Example: Using IFSFileOutputStream

The following example shows how to use the IFSFileOutputStream class:

```
                    // 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();
```

IFSTextFileOutputStream class:

IFSTextFileOutputStream has been deprecated and replaced by class IFSFileWriter.

The IFSTextFileOutputStream class represents a stream of character data being written to a file. The data supplied to the IFSTextFileOutputStream object is in a Java String object so the input is always Unicode. The IFSTextFileOutputStream object can convert the data to another CCSID as it is written to the file, however. The default behavior is to write Unicode characters to the file, but the Java program can set the target CCSID before the file is opened. In this case, the IFSTextFileOutputStream object converts the characters from Unicode to the specified CCSID before writing them to the file. If the data cannot be converted, an UnsupportedEncodingException is thrown.

Example: Using IFSTextFileOutputStream

The following example shows how to use IFSTextFileOutputStream:

```
                    // 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();
```

Related reference

"IFSFileWriter"
Use IFSFileWriter for writing character files in the integrated file system. IFSFileWriter is meant for writing streams of characters.

IFSFileWriter:

Use IFSFileWriter for writing character files in the integrated file system. IFSFileWriter is meant for writing streams of characters.

IFSFileWriter is the replacement of IFSTextFileOutputStream.

Example: Using IFSFileWriter

The following example illustrates the use of IFSFileWriter:

```
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();
```

Related information

IFSFileWriter Javadoc

IFSRandomAccessFile:

The IFSRandomAccessFile class represents a file on the server for reading and writing data.

IFSRandomAccessFile

The Java program can read and write data sequentially or randomly. As in IFSFile, methods exist in IFSRandomAccessFile that duplicate the methods in RandomAccessFile from the java.io package. In addition to these methods, IFSRandomAccessFile has additional methods specific to the iSeries server. Through IFSRandomAccessFile, a Java program can do the following:

- Open a file for read, write, or read/write access. The Java program can optionally specify the file sharing mode and the existence option.
- Read data at the current offset from the file.
- Write data at the current offset to the file.
- Get or set the current offset of the file.
- Close the file.

In addition to the methods in java.io RandomAccessFile, IFSRandomAccessFile gives the Java program the following options:

- Committing to disk bytes written.
- Locking or unlocking bytes in the file.
- Locking and unlocking bytes in the stream. See IFSKey for more information.
- Specifying a sharing mode when the file is opened. See sharing modes for more information.
- Specify the existence option when a file is opened. The Java program can choose one of the following:
  - Open the file if it exists; create the file if it does not.
  - Replace the file if it exists; create the file if it does not.
  - Fail the open if the file exists; create the file if it does not.
  - Open the file if it exists; fail the open if it does not.
  - Replace the file if it exists; fail the open if it does not.

Example: Using IFSRandomAccessFile

The following example shows how to use the IFSRandomAccessFile class to write four bytes at 1K intervals to a file.

```
                    // 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();
```

IFSFileDialog:

The IFSFileDialog class allows you to traverse the file system and select a file.

IFSFileDialog

This class uses the IFSFile class to traverse the list of directories and files in the integrated file system on the iSeries server. Methods on the class allow a Java program to set the text on the push buttons of the dialog and to set filters. Note that an IFSFileDialog class based on Swing 1.1 is also available.

You can set filters through the FileFilter class. If the user selects a file in the dialog, the getFileName() method can be used to get the name of the file that was selected. The getAbsolutePath() method can be used to get the path and name of the file that was selected.

Example: Using IFSFileDialog

The following example shows how to set up a dialog with two filters and to set the text on the push buttons of the dialog.

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

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

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

dialog.setFileFilter(filterList, 0);

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

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

IFSKey class:

If the Java program allows other programs access to a file at the same time, the Java program can lock bytes in the file for a period of time. During that time, the program has exclusive use of that section of the file.

When a lock is successful, the integrated file system classes return an IFSKey object. This object is supplied to the unlock() method to indicate which bytes to unlock. When the file is closed, the system unlocks all locks that are still on the file (the system does an unlock for every lock that the program did not unlock).

Example: Using IFSKey

The following example shows how to use the IFSKey class.

```
                    // 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();
```

File sharing mode:

The Java program can specify a sharing mode when a file is opened. The program either allows other programs to open the file at the same time or has exclusive access to the file.

The following example shows how to specify a file sharing mode.

```
                  // 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 provides a gateway to the iSeries integrated file system, for use when constructing javax.swing.JFileChooser objects.

JFileChooser is a standard Java way to build dialogs for navigating and choosing files.

Example: Using IFSSystemView

The following example demonstrates the use of IFSSystemView:

```
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());
 }
```

Related information

IFSSystemView Javadoc

IBM i NetServer

The IBM i NetServer class represents the NetServer service on a server. This class allows the user to query and modify the state and configuration of the NetServer.

IBM i NetServer replaces the NetServer class.

Related information

IBM i NetServer Javadoc

JavaApplicationCall

The JavaApplicationCall class provides you with the ability to have your client use the server JVM to run a Java program that resides on the server.

JavaApplicationCall

After establishing a connection to the server from the client, the JavaApplicationCall class lets you configure the following:

1. Set the CLASSPATH environment variable on the server with the setClassPath() method
2. Define your program's parameters with the setParameters() method
3. Run the program with run()
4. Send input from the client to the Java program. The Java program reads the input via standard input which is set with the sendStandardInString() method. You can redirect standard output and standard error from the Java program to the client via the getStandardOutString() and getStandardErrorString()

JavaApplicationCall is a class you call from your Java program. However, the IBM Toolbox for Java also provides utilities to call Java programs that reside on the server. These utilities are complete Java programs you can run from your workstation. See RunJavaApplication class for more information.

Example

The example in the JavaApplicationCall Javadoc reference documentation shows how to run a program on the server (that outputs ″Hello World!″) from the client:

JavaApplicationCall

JDBC classes

JDBC is an application programming interface (API) included in the Java platform that enables Java programs to connect to a wide range of databases.

For more information about JDBC and about IBM Toolbox for Java support for JDBC, including ongoing improvements, JDBC properties, and unsupported SQL types, see the following page:

"JDBC" on page 306

Supported interfaces

The following table lists the supported JDBC interfaces and the API required to use them:

Supported JDBC interface	API required
Blob provides access to binary large objects (BLOBs)	JDBC 2.1 core
CallableStatement runs SQL stored procedures	JDK 1.1
Clob provides access to character large objects (CLOBs)	JDBC 2.1 core

Supported JDBC interface	API required
Connection represents a connection to a specific database	JDK 1.1
ConnectionPool represents a pool of Connection objects.	JDBC 2.0 Optional Package
ConnectionPoolDataSource represents a factory for pooled AS400JDBCPooledConnection objects	JDBC 2.0 Optional Package
DatabaseMetaData provides information about the database as a whole.	JDK 1.1
DataSource represents a factory for database connections.	JDBC 2.0 Optional Package
Driver creates the connection and returns information about the driver version.	JDK 1.1
ParameterMetaData provides the ability to get information about the types and properties of the parameters in a PreparedStatement object	JDBC 3.0 API
PreparedStatement runs compiled SQL statements	JDK 1.1
ResultSet provides access to a table of data that is generated by running a SQL query or DatabaseMetaData catalog method	JDK 1.1
ResultSetMetaData provides information about a specific ResultSet	JDK 1.1
RowSet is a connected row set that encapsulates a ResultSet	JDBC 2.0 Optional Package
Savepoint provides finer grained control within transactions	JDBC 3.0 API
Statement runs SQL statements and obtains the results	JDK 1.1
XAConnection is a database connection which participates in global XA transactions	JDBC 2.0 Optional Package
XAResource is resource manager for use in XA transactions	JDBC 2.0 Optional Package

Examples

The following examples illustrate ways to use the IBM Toolbox for Java JDBC driver.

- "Example: Using JDBCPopulate to create and populate a table" on page 476
- "Example: Using JDBCQuery to query a table" on page 478

AS400JDBCBlob class:

You can use a AS400JDBCBlob object to access binary large objects (BLOBs), such as sound byte (.wav) files or image (.gif) files.

AS400JDBCBlob

The key difference between the AS400JDBCBlob class and the AS400JDBCBlobLocator class is where the blob is stored. With the AS400JDBCBlob class, the blob is stored in the database, which inflates the size of the database file. The AS400JDBCBlobLocator class stores a locator (think of it as a pointer) in the database file that points to where the blob is located.

With the AS400JDBCBlob class, the lob threshold property can be used. This property specifies the maximum large object (LOB) size (in kilobytes) that can be retrieved as part of a result set. LOBs that are larger than this threshold are retrieved in pieces using extra communication to the server. Larger LOB

thresholds reduce the frequency of communication to the server, but they download more LOB data, even if it is not used. Smaller lob thresholds may increase frequency of communication to the server, but they only download LOB data as it is needed. See JDBC properties for information about additional properties that are available.

Using the AS400JDBCBlob class, you can do the following:

- Return the entire blob as a stream of uninterpreted bytes
- Return part of the contents of the blob
- Return the length of the blob
- Create a binary stream to write to the blob
- Write a byte array to the blob
- Write all or a portion of byte array to the blob
- Truncate the blob

Examples

The following examples show how to use the AS400JDBCBlob class to read from a blob and update a blob:

Example: Using the AS400JDBCBlob class to read from a blob

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

Example: Using the AS400JDBCBlob class to update a blob

```
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 class

You can use a AS400JDBCBlobLocator object to access a binary large objects.

Using the AS400JDBCBlobLocator class, you can do the following:

- Return the entire blob as a stream of uninterpreted bytes
- Return part of the contents of the blob
- Return the length of the blob
- Create a binary stream to write to the blob
- Write a byte array to the blob
- Write all or a portion of byte array to the blob
- Truncate the blob

CallableStatement interface:

Use a CallableStatement object to run SQL stored procedures. The stored procedure being called must already be stored in the database. CallableStatement does not contain the stored procedure, it only calls the stored procedure.

CallableStatement

A stored procedure can return one or more ResultSet objects and can use IN parameters, OUT parameters, and INOUT parameters. Use Connection.prepareCall() to create new CallableStatement objects.

The CallableStatement object allows you to submit multiple SQL commands as a single group to a database through the use of batch support. You may get better performance by using batch support because processing a group of operations is usually faster than processing them one at a time. For more information about using batch support, see Enhancements to JDBC support.

CallableStatement allows you to get and set parameters and columns by name, although using the column index results in better performance.

Example: Using CallableStatement

The following example shows how to use the CallableStatement interface.

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

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

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

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

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

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

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

AS400JDBCClob class:

You can use a AS400JDBCClob object to access character large objects (CLOBs), such as large documents.

AS400JDBCClob

The key difference between the AS400JDBCClob class and the AS400JDBCClobLocator class is where the blob is stored. With the AS400JDBCClob class, the clob is stored in the database, which inflates the size of the database file. The AS400JDBCClobLocator class stores a locator (think of it as a pointer) in the database file that points to where the clob is located.

With the AS400JDBCClob class, the lob threshold property can be used. This property specifies the maximum large object (LOB) size (in kilobytes) that can be retrieved as part of a result set. LOBs that are larger than this threshold are retrieved in pieces using extra communication to the server. Larger LOB thresholds reduce the frequency of communication to the server, but they download more LOB data, even if it is not used. Smaller lob thresholds may increase frequency of communication to the server, but they only download LOB data as it is needed. See “IBM Toolbox for Java JDBC properties” on page 311 for information on additional properties that are available.

Using the AS400JDBCClob class, you can do the following:

- Return the entire clob as a stream of ASCII characters
- Return the contents of the clob as a character stream
- Return a part of the contents of the clob
- Return the length of the clob
- Create a Unicode character stream or an ASCII character stream to write to the clob
- Write a String to the clob
- Truncate the clob

Examples

The following examples show how to use the AS400JDBCClob class to read from a clob and update a clob:

Example: Using the AS400JDBCClob class to read from a clob

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

Example: Using the AS400JDBCClob class to update a clob

```
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 class

You can use a AS400JDBCClobLocator object to access character large objects (CLOBs).

Using the AS400JDBCClobLocator class, you can do the following:

- Return the entire clob as a stream of ASCII characters
- Return the entire clob as a character stream
- Return a part of the contents of the clob
- Return the length of the clob
- Create a Unicode character stream or an ASCII character stream to write to the clob
- Write a String to the clob
- Truncate the clob

AS400JDBCConnection class:

The AS400JDBCConnection class provides a JDBC connection to a specific DB2 UDB for iSeries database.

Use DriverManager.getConnection() to create new AS400JDBCConnection objects. For more information, see "Registering the JDBC driver" on page 69.

There are many optional properties that can be specified when the connection is created. Properties can be specified either as part of the URL or in a java.util.Properties object. See "IBM Toolbox for Java JDBC properties" on page 311 for a complete list of properties supported by the AS400JDBCDriver.

Note: A connection may contain at most 9999 open statements.

AS400JDBCConnection includes support for savepoints and statement-level holdability, and limited support for returning auto-generated keys. For more information about these and other enhancements, see "Enhancements to JDBC support for Version 5 Release 3" on page 308.

To use Kerberos tickets, set only the system name (and not the password) on your JDBC URL object. The user identity is retrieved through the Java Generic Security Services (JGSS) framework, so you also do not need to specify a user on your JDBC URL. You can set only one means of authentication in an AS400JDBCConnection object at a time. Setting the password clears any Kerberos ticket or profile token.

For more information, see the "AS400 class" on page 23 and J2SDK, v1.4 Security Documentation  .

Using the AS400JDBCConnection class, you can do the following:

- Create a statement (Statement, PreparedStatement, or CallableStatement objects)
- Create a statement that has a specific result set type and concurrency (Statement, PreparedStatement, or CallableStatement objects)
- Commit and rollback changes to the database and release database locks currently held
- Close the connection, closing server resources immediately instead of waiting for them to be automatically released
- Set holdability and get holdability for the Connection
- Set the transaction isolation and get the transaction isolation for the Connection
- Get the meta data for the Connection
- Set auto commit on or off
- Get the job identifier of the host server job that corresponds to the Connection

If you use JDBC 3.0 and connect to a server running i5/OS V5R2 or later, you can use AS400JDBCConnection to perform the following actions:

- Create a statement with a specific result set holdability (Statement, PreparedStatement, or CallableStatement object)
- Create a prepared statement that returns any auto-generated keys (when getGeneratedKeys() is called on the Statement object)
- Use savepoints, which offer more granular control over transactions:

- Set savepoints
- Rollback savepoints
- Release savepoints

AS400JDBCConnectionPool:

The AS400JDBCConnectionPool class represents a pool of AS400JDBCConnection objects that are available for use by a Java program as part of IBM Toolbox for Java support for the JDBC 2.0 Optional Package API.

You can use an AS400JDBCConnectionPoolDataSource to specify properties for the connections that are created in the pool, as in the following example.

You cannot change the connection pool data source after you have requested a connection and the pool is in use. To reset the connection pool data source, first call close() on the pool.

Return connections to an AS400JDBCConnectionPool by using close() on the AS400JDBCConnection object.

Note: When connections are not returned to the pool, the connection pool continues to grow in size and connections are not reused.

Set properties on the pool by using methods inherited from ConnectionPool. Some of the properties that you can set include:

- maximum number of connections to allow in the pool
- maximum lifetime of a connection
- maximum inactivity time of a connection.

You can also register AS400JDBCConnectionPoolDataSource objects by using a Java Naming and Directory Interface(TM) (JNDI) service provider. For more information on JNDI service providers, see IBM Toolbox for Java reference links.

Example: Using connection pool

The following example gets a connection pool data source from JNDI and uses it to create a connection pool with 10 connections:

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

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

     // Adds 10 connections to the pool that can be used by the
     // application (creates the physical database connections based on
     // the data source).
pool.fill(10);

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

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

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

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

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

DatabaseMetaData interface:

You can use a DatabaseMetaData object to obtain information about the database as a whole as well as catalog information.

The following example shows how to return a list of tables, which is a catalog function:

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

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

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

                  // Iterate through the ResultSet to get the values.

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

AS400JDBCDataSource class:

The AS400JDBCDataSource class represents a factory for iSeries database connections. The AS400JDBCConnectionPoolDataSource class represents a factory for AS400JDBCPooledConnection objects.

You can register either kind of data source object by using a Java Naming and Directory Interface (JNDI) service provider. For more information about JNDI service providers, see IBM Toolbox for Java reference links.

Examples

The following examples demonstrate ways to create and use AS400JDBCDataSource objects. The last two examples show how to register an AS400JDBCDataSource object with JNDI and then use the object returned from JNDI to obtain a database connection. Notice that even when using different JNDI service providers, the code is very similar.

Example: Creating an AS400JDBCDataSource object

The following example shows you how to create an AS400JDBCDataSource object and connect it to a database:

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

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

Example: Creating an AS400JDBCConnectionPoolDataSource object that can be used to cache JDBC connections

The following example shows how you can use an AS400JDBCConnectionPoolDataSource to cache JDBC connections.

```
// 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();
```

Example: Using JNDI service provider classes to store an AS400JDBCDataSource object

The following example shows how you can use JNDI service provider classes to store a DataSource object directly to the integrated file system on the server:

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

// Register the datasource with the Java Naming and Directory Interface (JNDI).
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
Context context = new InitialContext(env);
context.bind("jdbc/customer", dataSource);

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

Example: Using AS400JDBCDataSource objects and IBM SecureWay Directory classes with a Lightweight Directory Access Protocol (LDAP) directory server

The following examples shows how you can use IBM SecureWay® Directory classes to store an object to a Lightweight Directory Access Protocol (LDAP) directory server:

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

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

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

Registering the JDBC driver:

Before using JDBC to access data in a server database file, you need to register the JDBC driver for the IBM Toolbox for Java licensed program with the DriverManager. You can register the driver either by using a Java system property or by having the Java program register the driver:

- Register by using a system property

  Each virtual machine has its own method of setting system properties. For example, the Java command from the JDK uses the -D option to set system properties. To set the driver using system properties, specify the following:

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

- Register by using the Java program

  To load the IBM Toolbox for Java JDBC driver, add the following to the Java program before the first JDBC call:

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

  The IBM Toolbox for Java JDBC driver registers itself when it is loaded, which is the preferred way to register the driver. You can also explicitly register the IBM Toolbox JDBC driver by using the following:

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

The IBM Toolbox for Java JDBC driver does not require an AS400 object as an input parameter like the other IBM Toolbox for Java classes that get data from a server. An AS400 object is used internally, however, to manage default user and password caching. When a connection is first made to the server, the user may be prompted for user ID and password. The user has the option to save the user ID as the default user ID and add the password to the password cache. As in the other IBM Toolbox for Java functions, if the user ID and password are supplied by the Java program, the default user is not set and the password is not cached. See “Managing connections” on page 424 for information on managing connections.

Using the JDBC driver to connect to a database on the server

You can use the DriverManager.getConnection() method to connect to the server database. DriverManager.getConnection() takes a uniform resource locator (URL) string as an argument. The JDBC driver manager attempts to locate a driver that can connect to the database that is represented by the URL. When using the IBM Toolbox for Java driver, use the following syntax for the URL:

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

Note: Either systemName or defaultSchema can be omitted from the URL.

To use Kerberos tickets, set only the system name (and not the password) on your JDBC URL object. The user identity is retrieved through the Java Generic Security Services (JGSS) framework, so you also do not need to specify a user on your JDBC URL. You can set only one means of authentication in an AS400JDBCConnection object at a time. Setting the password clears any Kerberos ticket or profile token.

For more information, see “AS400 class” on page 23 and J2SDK, v1.4 Security Documentation .

Examples: Using the JDBC driver to connect to a server

Example: Using a URL in which a system name is not specified

This example results in the user being prompted to type in the name of the system to which he or she wants to connect.

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

Example: Connecting to the server database; no default schema or properties specified

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

Example: Connecting to the server database; default schema specified

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

Example: Connecting to the server database and using java.util.Properties to specify properties

The Java program can specify a set of JDBC properties either by using the java.util.Properties interface or by specifying the properties as part of the URL. See “IBM Toolbox for Java JDBC properties” on page 311 for a list of supported properties.

For example, to specify properties using the Properties interface, use the following code as an example:

```
                    // 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);
```

Example: Connecting to the server database and using a uniform resource locator (URL) to specify properties

```
                    // 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");
```

Example: Connecting to the server database and specifying user ID and password

```
                    // 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");
```

Example: Disconnecting from the database To disconnect from the server, use the close() method on the Connecting object. Use the following statement to close the connection created in the previous example:

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

AS400JDBCParameterMetaData class:

The AS400JDBCParameterMetaData class enables your programs to retrieve information about the properties of parameters in PreparedStatement and CallableStatement objects.

AS400JDBCParameterMetaData provides methods that allow you to perform the following actions:

- Get the class name of the parameter
- Get the number of parameters in the PreparedStatement
- Get the SQL type of the parameter
- Get the database-specific type name for the parameter
- Get the precision or the scale of the parameter

Example: Using AS400JDBCParameterMetaData

The following example shows one way to use AS400JDBCParameterMetaData to retrieve parameters from a dynamically generated PreparedStatement object:

```
// Get a connection from the driver.
Class.forName("com.ibm.as400.access.AS400JDBCDriver");
Connection connection =
   DriverManager.getConnection("jdbc:as400://myAS400", "myUserId", "myPassword");

// Create a prepared statement object.
PreparedStatement ps =
   connection.prepareStatement("SELECT STUDENTS FROM STUDENTTABLE WHERE STUDENT_ID= ?");

// Set a student ID into parameter 1.
ps.setInt(1, 123456);

// Retrieve the parameter meta data for the prepared statement.
ParameterMetaData pMetaData = ps.getParameterMetaData();

// Retrieve the number of parameters in the prepared statement.
// Returns 1.
int parameterCount = pMetaData.getParameterCount();

// Find out what the parameter type name of parameter 1 is.
// Returns INTEGER.
String getParameterTypeName = pMetaData.getParameterTypeName(1);
```

PreparedStatement interface:

You can use a PreparedStatement object when an SQL statement is going to be run many times. An SQL statement can be precompiled. A ″prepared.″ statement is an SQL statement that has been precompiled. This approach is more efficient than running the same statement multiple times using a Statement object, which compiles the statement each time it is run. In addition, the SQL statement contained in a PreparedStatement object may have one or more IN parameters. Use Connection.prepareStatement() to create PreparedStatement objects.

The PreparedStatement object allows you to submit multiple SQL commands as a single group to a database through the use of batch support. You may improve performance by using batch support because processing a group of operations is typically faster than processing them one at a time. For more information about using batch support, see Enhancements to JDBC support.

Example: Using PreparedStatement

The following example shows how to use the PreparedStatement interface.

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

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

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

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

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

ResultSet class:

You can use a ResultSet object to access a table of data that was generated by running a query. The table rows are retrieved in sequence. Within a row, column values can be accessed in any order.

The data stored in ResultSet is retrieved by using the various get methods, depending on the type of data being retrieved. The next() method is used to move to the next row.

ResultSet allows you to get and update columns by name, although using the column index results improves performance.

Cursor movement

A cursor, which is an internal pointer, is used by a result set to point the row in the result set that is being accessed by the Java program.

The performance of the getRow() method has been improved. Before V5R2, using ResultSet.last(), ResultSet.afterLast(), and ResultSet.absolute() with a negative value made the current row number not available. The previous restrictions are lifted, which makes the getRow() method fully functional.

JDBC 2.0 and later JDBC specifications provide additional methods for accessing specific positions within a database:

Scrollable cursor positions	
absolute	isFirst
afterLast	isLast
beforeFirst	last
first	moveToCurrentRow
getRow	moveToInsertRow
isAfterLast	previous
isBeforeFirst	relative

Scrolling capabilities

If a result set is created by executing a statement, you can move (scroll) backward (last-to-first) or forward (first-to-last) through the rows in a table.

A result set that supports this movement is called a scrollable result set. Scrollable result sets also support absolute positioning. Absolute positioning allows you to move directly to a row by specifying its position in the result set.

With JDBC 2.0 and later JDBC specifications, you have two additional scrolling capabilities available to use when working with the ResultSet class: scroll-insensitive and scroll-sensitive result sets.

A scroll-insensitive result set is not usually sensitive to changes that are made while it is open, while the scroll-sensitive result set is sensitive to changes.

Note: IBM iSeries Server only allows read-only access for scrollable insensitive cursors. IBM Toolbox for Java supports a scroll-insensitive cursor if the result set concurrency is read-only. If the result set type is specified as insensitive and the concurrency is specified as updateable, the result set type changes to sensitive and issues a warning to you.

Updateable result sets

In your application, you can use result sets that use either read-only concurrency (no updates can be made to the data) or updateable concurrency (allows updates to the data and uses database write locks to control access to the same data item by different transactions). In an updateable result set, rows can be updated, inserted, and deleted. Numerous update methods are available for you to use in your program, for example:

- Update ASCII stream
- Update Big Decimal
- Update binary stream

See Method Summary for a complete listing of the update methods available through the ResultSet interface.

Example: Updatable result sets

The following example shows how to use a result set that allows updates to the data (update concurrency) and allows changes to be made to the result set while it is open (scroll sensitive).

```
                    // 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

The ResultSetMetaData interface determines the types and properties of the columns in a ResultSet.

When connecting to a server running i5/OS V5R2 or later, using the extended metadata property enables you to increase the accuracy of the following ResultSetMetaData methods:

- getColumnLabel(int)
- isReadOnly(int)
- isSearchable(int)
- isWriteable(int)

Additionally, setting this property to true enables support for the ResultSetMetaData.getSchemaName(int) method. Be aware that using the extended metadata property may degrade performance because it requires retrieving more information from the server.

AS400JDBCRowSet class:

The AS400JDBCRowSet class represents a connected rowset that encapsulates a JDBC result set. The methods on AS400JDBCRowSet are very similar to those of the AS400JDBCResultSet. The database connection is maintained while in use.

You can use an instance of AS400JDBCDataSource or AS400JDBCConnectionPoolDataSource to create the connection to the database that you want to use to access the data for the AS400JDBCRowSet.

Examples

The following examples show how you can use the AS400JDBCRowSet class.

Example: Creating, populating, and updating an AS400JDBCRowSet object

```
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();
}
```

Example: Creating and populating an AS400JDBCRowSet object, while getting the data source from JNDI

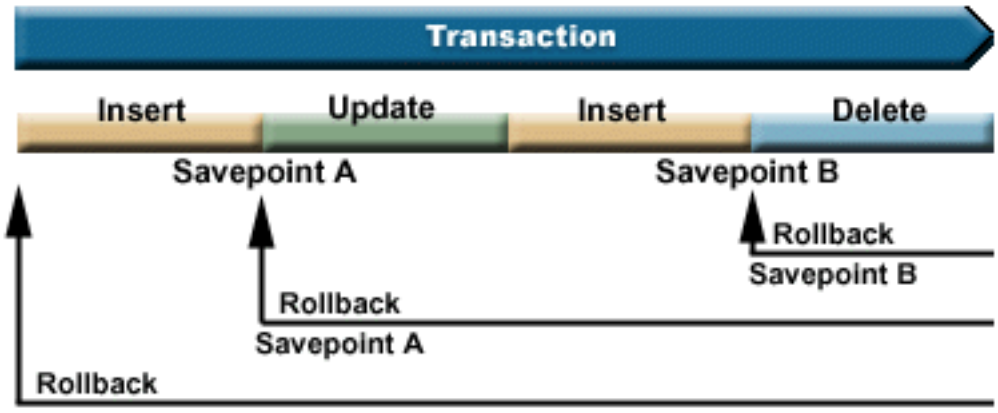
```
// Get the data source that is registered in JNDI (assumes JNDI environment is set).
Context context = new InitialContext();
AS400JDBCDataSource dataSource = (AS400JDBCDataSource) context.lookup("jdbc/customer");

AS400JDBCRowSet rowset = new AS400JDBCRowSet();
// Establish connection by setting the data source name.
rowset.setDataSourceName("jdbc/customer");
rowset.setUsername("myuser");
rowset.setPassword("myPasswd");

// Set the prepared statement and initialize the parameters.
rowset.setCommand("SELECT * FROM MYLIBRARY.MYTABLE WHERE STATE = ? AND BALANCE > ?");
rowset.setString(1, "MINNESOTA");
```

```
rowset.setDouble(2, MAXIMUM_LIMIT);

// Populate the rowset.
rowset.execute();
```

AS400JDBCSavepoint class:

The AS400JDBCSavepoint class represents a logical breaking point in a transaction. Using savepoints gives you more granular control over which changes are affected when you roll back a transaction.

Figure 1: Using savepoints to control rollbacks in a transaction



For example, Figure 1 shows a transaction that includes two savepoints, A and B. Rolling back the transaction to either savepoint undoes (or reverses) only those changes from the point a rollback is called to the savepoint. This prevents having to undo all the changes in the entire transaction. Note that once you rollback to savepoint A, you cannot later rollback to savepoint B. You cannot access savepoint B after work is rolled back past it.

Example: Using savepoints

In this scenario, assume that your application updates student records. At the end of updating a certain field in every student record, you perform a commit. Your code detects a particular error associated with updating this field and rolls back the work done when this error occurs. You know that this particular error affects only the work performed on the current record.

So, you set a savepoint between each update of student records. Now, when this error occurs, you rollback only the last update in the student table. Instead of having to roll back a large amount of work, you can now roll back only a small amount of work.

The following example code helps illustrate how you can use savepoints. The example assumes that the student ID for John is 123456 and the student ID for Jane is 987654.

```
// 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();
```

Considerations and restrictions

Using savepoints requires that you be aware of the following considerations and restrictions:

Considerations

IBM Toolbox for Java follows database rules regarding how rollbacks affect cursors and retained locks. For example, when you set the connection option to keep cursors open after a traditional rollback, cursors also remain open after a rollback to a savepoint. In other words, when a rollback request happens involving savepoints, IBM Toolbox for Java does not move or close the cursor when the underlying database does not support this.

Using a savepoint to roll back a transaction undoes only the actions performed from the point where you start the roll back to the savepoint. Actions performed before that savepoint remain. As in the previous example, be aware that you can commit a transaction that includes work performed before a particular savepoint but does not include work performed after the savepoint.

All savepoints are released and become invalid when the transaction is committed or when the entire transaction is rolled back. You can also release savepoints by calling Connection.releaseSavepoint().

Restrictions

The following restrictions apply when using savepoints:

- Named savepoints must be unique.
- You cannot reuse a savepoint name until the savepoint is released, committed, or rolled back.
- Auto-commit must be set to 'OFF' for savepoints to be valid. You can set auto-commit 'OFF' by using `Connection.setAutoCommit(false)`. Enabling auto-commit when using savepoints throws an exception.
- Savepoints are not valid across XA connections. Using an XA connection with savepoints throws an exception.
- Your server must be running i5/OS Version 5 Release 2 or later. Using savepoints when connecting (or already connected) to a server running V5R1 or earlier version of i5/OS throws an exception.

Long description of Figure 1: Using savepoints to control rollbacks in a transaction (rzahh586.gif):

found in IBM Toolbox for Java: AS400JDBCSavepoint class

This figure illustrates how to use savepoints to control rollbacks in a transaction.

Description

The figure is composed of the following:

- A blue horizontal arrow, pointing to the right, labeled 'Transaction.' The Transaction arrow represents a linear transaction that begins on the left and ends on the right.
- Below the Transaction arrow is a multicolored bar of equal length to the Transaction arrow. The bar is divided into four colored sections that, from left to right, represent the separate actions that constitute the transaction. Under the bar are two labels that represent savepoints in the transaction.
- Below the multicolored bar are three arrows positioned one above the other. The arrows point left, and each one represents rolling back the transaction to a different point.

The Transaction arrow represents a transaction that begins on the left and ends on the right. The transaction is composed of a series of separate actions (the different sections of the multicolored bar). From left to right, the colored sections represent:

- The first action (a tan section) labeled 'Insert'
- The second action (a green section) labeled 'Update'
- The third action (another tan section) labeled 'Insert'
- The fourth and final action (a blue section) labeled 'Delete'

Labels below the multicolored bar represent savepoints. The point where the first action ends and the second begins is labeled 'Savepoint A.' The point where the third action ends and the fourth begins is labeled 'Savepoint B.'

Arrows below the multicolored bar point to the left and represent how the savepoints affect rolling back the transaction:

- The first arrow points to Savepoint B. Rolling back the transaction to Savepoint B reverses only the final action (the blue section labeled 'Delete')
- The second arrow points to Savepoint A. Rolling back the transaction to Savepoint A reverses the second through fourth actions (the green section labeled 'Update,' the second tan section labeled 'Insert,' and the blue section labeled 'Delete')
- The final arrow points to the very beginning of the transaction. Completely rolling back the transaction reverses all of the separate actions

Running SQL statements with Statement objects:

Use a Statement object to run an SQL statement and optionally obtain the ResultSet produced by it.

PreparedStatement inherits from Statement, and CallableStatement inherits from PreparedStatement. Use the following Statement objects to run different SQL statements:

- "Statement interface" on page 79: Runs a simple SQL statement that has no parameters.
- "PreparedStatement interface" on page 72 - Runs a precompiled SQL statement that may or may not have IN parameters.
- "CallableStatement interface" on page 63 - Runs a call to a database stored procedure. A CallableStatement may or may not have IN, OUT, and INOUT parameters.

The Statement object allows you to submit multiple SQL commands as a single group to a database through the use of batch support. You may improve performance by using batch support because processing a group of operations is typically faster than processing them one at a time. For more information about using batch support, see Enhancements to JDBC support.

When using batch updates, typically you turn off auto-commit. Turning off auto-commit allows your program to determine whether to commit the transaction if an error occurs and not all of the commands have executed. In JDBC 2.0 and later JDBC specifications, a Statement object can keep track of a list of commands that can be successfully submitted and executed together in a group. When this list of batch commands is executed by the executeBatch() method, the commands are executed in the order in which they were added to the list.

AS400JDBCStatement provides methods that enable you to perform many actions, including the following:

- Execute different kinds of statements
- Retrieve the values for different parameters of the Statement object, including:
  - The connection
  - Any auto-generated keys created as a result of executing the Statement
  - The fetch size and fetch direction
  - The maximum field size and maximum row limit
  - The current result set, the next result set, the type of result set, the result set concurrency, and the result set cursor holdability
- Add an SQL statement to the current batch
- Run the current batch of SQL statements

Statement interface

Use Connection.createStatement() to create new Statement objects.

The following example shows how to use a Statement object.

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

                    // Create a Statement object.
Statement s = c.createStatement();

                    // Run an SQL statement that creates
                    // a table in the database.
s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

                    // Run an SQL statement that inserts
                    // a record into the table.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('DAVE', 123)");

                    // Run an SQL statement that inserts
                    // a record into the table.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('CINDY', 456)");

                    // Run an SQL query on the table.
ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

                    // Close the Statement and the
                    // Connection.
s.close();
c.close();
```

JDBC XA Distributed Transaction Management:

The JDBC XA distributed transaction management classes enable you to use the IBM Toolbox for Java JDBC driver within a distributed transaction. Using the XA classes to enable the IBM Toolbox for Java JDBC driver allows it to participate in transactions that span multiple data sources.

Typically, XA distributed transaction management classes are used and controlled directly by a transaction manager, which is separate from the JDBC driver. The distributed transaction management interfaces are defined as part of the JDBC 2.0 Optional Package and the Java Transaction API (JTA). Both are available from Sun as jar files. The distributed transaction management interfaces are also supported in the JDBC 3.0 API, which is bundled with the Java 2 Platform, Standard Edition, version 1.4.

For more information, see the Sun Web sites for JDBC and the JTA.

Use the following objects to enable the IBM Toolbox for Java JDBC driver to participate in XA distributed transactions:

- AS400JDBCXADataSource - A factory for AS400JDBCXAConnection objects. This is a subclass of AS400JDBCDataSource.
- AS400JDBCXACConnection - A pooled connection object that provides hooks for connection pool management and XA resource management.
- AS400JDBCXAResource - A resource manager for use in XA transaction management.

Note: Prior to V5R3, the database host server used XA APIs for Job Scoped Locks (XA model). In V5R3 and subsequent releases, the database host server uses XA APIs for Transaction Scoped Locks (NTS model) for all MTS functions. For more information about how these APIs differ, see XA APIs.

Example: Using XA classes

The following example shows simple usage of the XA classes. Keep in mind that the details would be filled in with work using other data sources. This type of code usually appears within a transaction manager.

```
// Create an XA data source for making the XA connection.
AS400JDBCXADataSource xaDataSource = new AS400JDBCXADataSource("myAS400");
xaDataSource.setUser("myUser");
xaDataSource.setPassword("myPasswd");

// Get an XAConnection and get the associated XAResource.
// This provides access to the resource manager.
XAConnection xaConnection = xaDataSource.getXAConnection();
XAResource xaResource = xaConnection.getXAResource();

// Generate a new Xid (this is up to the transaction manager).
Xid xid = ...;

// Start the transaction.
xaResource.start(xid, XAResource.TMNOFLAGS);

// ...Do some work with the database...

// End the transaction.
xaResource.end(xid, XAResource.TMSUCCESS);

// Prepare for a commit.
xaResource.prepare(xid);

// Commit the transaction.
xaResource.commit(xid, false);

// Close the XA connection when done.  This implicitly
// closes the XA resource.
xaConnection.close();
```

Jobs classes

The IBM Toolbox for Java Jobs classes (in the access package) allow a Java program to retrieve and change job information.

Note: Toolbox for Java also provides resource classes that present a generic framework and consistent programming interface for working with various iSeries objects and lists. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application. The resource classes for working with jobs are RJob, RJobList, and RJobLog.

Use the Jobs classes to work with the following type of job information:

- Date and Time Information

- Job Queue
- Language Identifiers
- Message Logging
- Output Queue
- Printer Information

The job classes in the access package are as follows:
- Job - retrieves and changes iSeries job information
- JobList - retrieves a list of iSeries jobs
- JobLog - represents the job log of an iSeries

Examples

The following examples show some of the ways you can use the Job, JobList, and JobLog classes. The first example shows one way to use a cache with the Job class. Links to other examples immediately follow the sample code.

Note: Read the Code example disclaimer for important legal information.

Example: Using a cache when setting a value and getting a value

```
try {

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

    // Constructs a Job object
    Job job = new Job(as400,"QDEV002");

    // Gets job information
    System.out.println("User of this job :" + job.getUser());
    System.out.println("CPU used :" + job.getCPUUsed();
    System.out.println("Job enter system date : " + job.getJobEnterSystemDate());

    // Sets cache mode
    job.setCacheChanges(true);

    // Changes will be store in the cache.
    job.setRunPriority(66);
    job.setDateFormat("*YMD");

    // Commit changes. This will change the value on the iSeries.
    job.commitChanges();

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

The following examples show how to list the jobs belonging to a specific user, list jobs with the job status information, and display the messages in a job log:

"Example: Using JobList to list job identification information" on page 481

"Example: Using JobList to get a list of jobs" on page 483

"Example: Using JobLog to display messages in the job log" on page 486

Job class:

The Job class (in the access package) allows a Java program to retrieve and change server job information.

Note: IBM Toolbox for Java also provides resource classes that present a generic framework and consistent programming interface for working with various objects and lists on your iSeries server. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application. The resource classes for working with jobs are RJob, RJobList, and RJobLog.

The following type of job information can be retrieved and changed with the Job class:

- Job queues
- Output queues
- Message logging
- Printer device
- Country or region identifier
- Date format

The job class also allows the ability to change a single value at a time, or cache several changes using the setCacheChanges(true) method and committing the changes using the commitChanges() method. If caching is not turned on, you do not need to do a commit.

Example

For a code example, see the Javadoc reference documentation for the Job class. The example shows how to set and get values to and from the cache in order to set the run priority with the setRunPriority() method and set the date format with the setDateFormat() method:

Job

JobList class:

You can use JobList class (in the access package) to list iSeries jobs.

Note: IBM Toolbox for Java also provides resource classes that present a generic framework and consistent programming interface for working with various iSeries objects and lists. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application. The resource classes for working with jobs are RJob, RJobList, and RJobLog.

With the JobList class, you can retrieve the following:

- All jobs
- Jobs by name, job number, or user

Use the getJobs() method to return a list of iSeries jobs or getLength() method to return the number of jobs retrieved with the last getJobs().

Example: Using JobList

The following example lists all active jobs on the system:

```
                    // Create an AS400 object. List the
                    // jobs on this iSeries.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Create the job list object.
```

```
JobList jobList = new JobList(sys);

                  // Get the list of active jobs.
Enumeration list = jobList.getJobs();

                  // For each active job on the system
                  // print job information.
while (list.hasMoreElements())
{
    Job j = (Job) list.nextElement();

    System.out.println(j.getName() + "." +
                       j.getUser() + "." +
                       j.getNumber());
}
```

JobLog class:

The JobLog class (in the access package) retrieves messages in the job log of a server job by calling getMessages().

Note: IBM Toolbox for Java also provides resource classes that present a generic framework and consistent programming interface for working with various iSeries objects and lists. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application. The resource classes for working with jobs are RJob, RJobList, and RJobLog.

Example: Using JobLog

The following example prints all messages in the job log for the specified user:

```
                  // ... Setup work to create an AS400
                  // object and a jobList object has
                  // already been done

                  // Get the list of active jobs on
                  // the iSeries
Enumeration list = jobList.getJobs();

                  // Look through the list to find a
                  // job for the specified user.
while (list.hasMoreElements())
{
   Job j = (Job) list.nextElement();

   if (j.getUser().trim().equalsIgnoreCase(userID))
   {
                  // A job matching the current user
                  // was found. Create a job log
                  // object for this job.
      JobLog jlog = new JobLog(system, j.getName(), j.getUser(), j.getNumber());

                  // Enumerate the messages in the job
                  // log then print them.
      Enumeration messageList = jlog.getMessages();

      while (messageList.hasMoreElements())
      {
          AS400Message message = (AS400Message) messageList.nextElement();
          System.out.println(message.getText());
      }

   }
}
```

Message classes

AS400Message

AS400Message object allows the Java program to retrieve an i5/OS message that is generated from a previous operation (for example, from a command call). From a message object, the Java program can retrieve the following:

- The iSeries library and message file that contain the message
- The message ID
- The message type
- The message severity
- The message text
- The message help text

The following example shows how to use the AS400Message object:

Note: Read the Code example disclaimer for important legal information.

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

Examples: Using message lists

The following examples show how you can use message lists with CommandCall and ProgramCall.

- "Example: Using CommandCall" on page 446
- "Example: Using ProgramCall" on page 498

QueuedMessage

The QueuedMessage class extends the AS400Message class.

Note: Toolbox for Java also provides resource classes that present a generic framework and consistent programming interface for working with various iSeries objects and lists. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application. The resource class for working with queued messages is RQueuedMessage.

The QueuedMessage class accesses information about a message on an iSeries message queue. With this class, a Java program can retrieve:

- Information about where a message originated, such as program, job name, job number, and user
- The message queue
- The message key

- The message reply status

The following example prints all messages in the message queue of the current (signed-on) user:

Note: Read the Code example disclaimer for important legal information.

```
                // The message queue is on this iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

                // Create the message queue object.
                // This object will represent the
                // queue for the current user.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                // Get the list of messages currently
                // in this user's queue.
Enumeration e = queue.getMessages();

                // Print each message in the queue.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

MessageFile

The MessageFile class allows you to receive a message from an iSeries message file. The MessageFile class returns an AS400Message object that contains the message. Using the MessageFile class, you can do the following:

- Return a message object that contains the message
- Return a message object that contains substitution text in the message

The following example shows how to retrieve and print a message:

Note: Read the Code example disclaimer for important legal information.

```
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

The MessageQueue class allows a Java program to interact with an iSeries message queue.

Note: Toolbox for Java also provides resource classes that present a generic framework and consistent programming interface for working with various iSeries objects and lists. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application. The resource class for working with message queues is RMessageQueue.

The MessageQueue class acts as a container for the QueuedMessage class. The getMessages() method, in particular, returns a list of QueuedMessage objects. The MessageQueue class can do the following:

- Set message queue attributes
- Get information about a message queue
- Receive messages from a message queue
- Send messages to a message queue
- Reply to messages

The following example lists messages in the message queue for the current user:

Note: Read the Code example disclaimer for important legal information.

```
                  // The message queue is on this iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

                  // Create the message queue object.
                  // This object will represent the
                  // queue for the current user.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                  // Get the list of messages currently
                  // in this user's queue.
Enumeration e = queue.getMessages();

                  // Print each message in the queue.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

NetServer

NetServer has been deprecated and replaced by class ISeriesNetServer.

The NetServer class represents the NetServer service on an iSeries server. NetServer objects allow you to query and modify the state and configuration of the NetServer service.

For example, you can use the NetServer class to:

- Start or stop the NetServer
- Get a list of all current file shares and print shares
- Get a list of all current sessions
- Query and change attribute values (using methods inherited from ChangeableResource)

Note: In order to use the NetServer class, you need a server user profile that has *IOSYSCFG authority.

The NetServer class is an extension of ChangeableResource and Resource, so it provides a collection of ″attributes″ to represent the various NetServer values and settings. You query or change the attributes in order to access or change the configuration of your NetServer. Some of the NetServer attributes are:

- NAME
- NAME_PENDING
- DOMAIN
- ALLOW_SYSTEM_NAME
- AUTOSTART
- CCSID
- WINS_PRIMARY_ADDRESS

Pending attributes

Many of the NetServer attributes are pending (for example, NAME_PENDING). Pending attributes represent NetServer values that take effect the next time you start (or restart) the NetServer on the server.

When you have a pair of related attributes and one attribute is pending while the other is nonpending:

- The pending attribute is read/write, so you can change it
- The nonpending attribute is read-only, so you can query it but you cannot change it

Other NetServer classes

Related NetServer classes allow you to get and set detailed information about specific connections, sessions, file shares, and print shares:

- NetServerConnection: Represents a NetServer connection
- NetServerFileShare: Represents a NetServer file server share
- NetServerPrintShare: Represents a NetServer print server share
- NetServerSession: Represents a NetServer session
- NetServerShare: Represents a NetServer share

Example: Using a NetServer object to change the name of the NetServer

Note: Read the Code example disclaimer for important legal information.

```
// Create a system object to represent the iSeries server.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWD");

// Create an object with which to query and modify the NetServer.
NetServer nServer = new NetServer(system);

// Set the "pending name" to NEWNAME.
nServer.setAttributeValue(NetServer.NAME_PENDING, "NEWNAME");

// Commit the changes.  This sends the changes to the server.
nServer.commitAttributeChanges();

// The NetServer name will get set to NEWNAME the next time the NetServer
// is ended and started.
```

Permission classes

The permission classes allow you to get and set object authority information. Object authority information is also known as permission. The Permission class represents a collection of many users' authority to a specific object. The UserPermission class represents a single user's authority to a specific object.

Permission class

The Permission class allows you to retrieve and change object authority information. It includes a collection of many users who are authorized to the object. The Permission object allows the Java program to cache authority changes until the commit() method is called. Once the commit() method is called, all changes made up to that point are sent to the server. Some of the functions provided by the Permission class include:

- addAuthorizedUser(): Adds an authorized user.
- commit(): Commits the permission changes to the server.
- getAuthorizationList(): Returns the authorization list of the object.
- getAuthorizedUsers(): Returns an enumeration of authorized users.
- getOwner(): Returns the name of the object owner.
- getSensitivityLevel(): Returns the sensitivity level of the object.
- getType(): Returns the object authority type (QDLO, QSYS, or Root).
- getUserPermission(): Returns the permission of a specific user to the object.
- getUserPermissions(): Returns an enumeration of permissions of the users to the object.
- setAuthorizationList(): Sets the authorization list of the object.
- setSensitivityLevel(): Sets the sensitivity level of the object.

Example: Using Permission

Note: Read the Code example disclaimer for important legal information.

The following example shows you how to create a permission and add an authorized user to an object.

```
// 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 class

The UserPermission class represents the authority of a single, specific user. UserPermission has three subclasses that handle the authority based on the object type:

UserPermission class	Description
DLOPermission	Represents a user's authority to Document Library Objects (DLOs), which are stored in QDLS.
QSYSPermission	Represents a user's authority to objects stored in QSYS.LIB and contained in the server.
RootPermission	Represents a user's authority to objects contained in the root directory structure. RootPermissions objects are those objects not contained in QSYS.LIB or QDLS.

The UserPermission class allows you to do the following:

- Determine if the user profile is a group profile
- Return the user profile name
- Indicate whether the user has authority
- Set the authority of authorization list management

Example: Using UserPermission

Note: Read the Code example disclaimer for important legal information.

The following example shows you how to retrieve the users and groups that have permission on an object and print them out one at a time.

```
// Create a system object.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Represent the permissions to an object on the system, such as a library.
Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

// Retrieve the various users/groups that have permissions set on that object.
Enumeration enum = objectInQSYS.getUserPermissions();
while (enum.hasMoreElements())
{
  // Print out the user/group profile names one at a time.
  UserPermission userPerm = (UserPermission)enum.nextElement();
  System.out.println(userPerm.getUserID());
}
```

DLOPermission class:

DLOPermission is a subclass of UserPermission. DLOPermission allows you to display and set the authorities a user has (called permissions) to a document library object (DLO).

One of the following authority values is assigned to each user.

Authority value	Description
*ALL	The user can perform all operations except those operations that are controlled by authorization list management.
*AUTL	The authorization list is used to determine the authority for the document.
*CHANGE	The user can change and perform basic functions on the object.
*EXCLUDE	The user cannot access the object.
*USE	The user has object operational authority, read authority, and execute authority.

You must use one of the following methods to change or determine the user's authority:

- Use getDataAuthority() to display the authority value of the user
- Use setDataAuthority() to set the authority value of the user

After setting permissions, it is important that you use the commit() method from the Permissions class to send the changes to the server.

For more information about permissions and authorities, see Chapter 5: Resource Security in the **iSeries Security Reference** .

Example: Using DLOPermission

The following example shows how to retrieve and print the DLO permissions, including the user profiles for each permission.

```
// Create a system object.

AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");
// Represent the permissions to a DLO object.
Permission objectInQDLS = new Permission(sys, "/QDLS/MyFolder");

// Print the object pathname and retrieve its permissions.
System.out.println("Permissions on " + objectInQDLS.getObjectPath() + " are as follows:");
Enumeration enum = objectInQDLS.getUserPermissions();
while (enum.hasMoreElements())
{
  // For each of the permissions, print out the user profile name
  // and that user's authorities to the object.
  DLOPermission dloPerm = (DLOPermission)enum.nextElement();
  System.out.println(dloPerm.getUserID() + ": " + dloPerm.getDataAuthority());
}
```

QSYSPermission:

QSYSPermission is a subclass of the UserPermission class. QSYSPermission allows you to display and set the permission a user has for an object in the traditional iSeries library structure stored in QSYS.LIB. You can set authority for an object stored in QSYS.LIB by setting a system-defined authority value or by setting the individual object and data authorities.

The following table lists and describes the valid system-defined authority values:

System-defined authority value	Description
*ALL	The user can perform all operations except those operations that are controlled by authorization list management.
*AUTL	The authorization list is used to determine the authority for the document.
*CHANGE	The user can change and perform basic functions on the object.
*EXCLUDE	The user cannot access the object.
*USE	The user has object operational authority, read authority, and execute authority.

Each system-defined authority value actually represents a combination of the individual object authorities and data authorities. The following table illustrates the relationships of system-defined authorities to the individual object and data authorities:

Table 1. **Y** *refers to those authorities that can be assigned. n refers to those authorities that cannot be assigned.*

System-defined authority	Object authority					Data authority				
	Opr	Mgt	Exist	Alter	Ref	Read	Add	Upd	Dlt	Exe
All	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
Change	**Y**	n	n	n	n	**Y**	**Y**	**Y**	**Y**	**Y**
Exclude	n	n	n	n	n	n	n	n	n	n
Use	**Y**	n	n	n	n	**Y**	n	n	n	**Y**
Autl	Only valid with user (*PUBLIC) and a specified authorization list that determines the individual object and data authorities.									

Specifying a system-defined authority automatically assigns the appropriate individual authorities. Likewise, specifying various individual authorities changes the appropriate individual authority values. When a combination of individual object authorities and data authorities does not map to a single system-defined authority value, then the single value becomes ″User Defined.″

Use the getObjectAuthority() method to display the current system-defined authority. Use the setObjectAuthority() method to set the current system-defined authority using a single value.

Use the appropriate set method to set individual object authority values on or off:

- setAlter()
- setExistence()
- setManagement()
- setOperational()
- setReference()

Use the appropriate set method to set individual data authority values on or off:

- setAdd()
- setDelete()
- setExecute()
- setRead()

- setUpdate()

For more information about the different authorities, see Chapter 5: Resource Security in the **iSeries Security Reference**  . For information about using iSeries CL commands to grant and edit object authorities, see the iSeries CL commands Grant Object Authority (GRTOBJAUT) and Edit Object Authority (EDTOBJAUT).

Example

This example shows you how to retrieve and print the permissions for a QSYS object.

```
// Create a system object.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Represent the permissions to a QSYS object.
Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

// Print the object pathname and retrieve its permissions.
System.out.println("Permissions on "+objectInQSYS.getObjectPath()+" are as follows:");
Enumeration enum = objectInQSYS.getUserPermissions();
while (enum.hasMoreElements())
{
  // For each of the permissions, print out the user profile name
  // and that user's authorities to the object.
  QSYSPermission qsysPerm = (QSYSPermission)enum.nextElement();
  System.out.println(qsysPerm.getUserID()+": "+qsysPerm.getObjectAuthority());
}
```

RootPermission:

RootPermission is a subclass of the UserPermission class. The RootPermission class allows you to display and set the permissions for the user of an object contained in the root directory structure.

An object on the root directory structure can set the data authority or the object authority. You can set the data authority to the values listed in the following table. Use the getDataAuthority() method to to display the current values and the setDataAuthority() method to set the data authority.

The following table lists and describes the valid data authority values:

Data authority value	Description
*none	The user has no authority to the object.
*RWX	The user has read, add, update, delete, and execute authorities.
*RW	The user has read, add, and delete authorities.
*RX	The user has read and execute authorities.
*WX	The user has add, update, delete, and execute authorities.
*R	The user has read authority.
*W	The user has add, update, and delete authorities.
*X	The user has execute authority.
*EXCLUDE	The user cannot access the object.
*AUTL	The public authorities on this object come from the authorization list.

The object authority can be set to one or more of the following values: alter, existence, management, or reference. You can use the setAlter(), setExistence(), setManagement(), or setReference() methods to set the values on or off.

After setting either the data authority or the object authority of an object, it is important that you use the commit() method from the Permissions class to send the changes to the server.

For more information about the different authorities, see Chapter 5: Resource Security in the **iSeries Security Reference** .

Example

This example shows you how to retrieve and print the permissions for a root object.

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

Print classes

Print objects include spooled files, output queues, printers, printer files, writer jobs, and Advanced Function Printing™ (AFP) resources, which include fonts, form definitions, overlays, page definitions, and page segments. AFP resources are accessible only on Version 3 Release 7 (V3R7) and later versions of i5/OS. (Trying to open an AFPResourceList to a system that is running an earlier version than V3R7 generates a RequestNotSupportedException exception.)

The IBM Toolbox for Java classes for print objects are organized on a base class, PrintObject, and on a subclass for each of the six types of print objects. The base class contains the methods and attributes common to all server print objects. The subclasses contain methods and attributes specific to each subtype.

Use the print classes for the following tasks:

- Working with server print objects:
  - PrintObjectList class - use for listing and working with server print objects. (Print objects include spooled files, output queues, printers, Advanced Function Printing (AFP) resources, printer files, and writer jobs)
  - PrintObject base class - use for working with print objects
- Retrieving PrintObject attributes
- Creating new server spooled files using the SpooledFileOutputStream class (use for EBCDIC-based printer data)
- Generating SNA Character Stream (SCS) printer data streams
- Reading spooled files and AFP resources using the PrintObjectInputStream
- Reading spooled files using PrintObjectPageInputStream and PrintObjectTransformedInputStream
- Copying spooled files
- Viewing Advanced Function Printing (AFP) and SNA Character Stream (SCS) spooled files

Examples

- Example: Creating spooled files shows how to create a spooled file on a server from an input stream
- Example: Creating SCS spooled files shows how to generate a SCS data stream using the SCS3812Writer class, and how to write the stream to a spooled file on the server
- Example: Reading spooled files shows how to use PrintObjectInputStream to read an existing server spooled file
- Example: Reading and transforming spooled files shows how to use PrintObjectPageInputStream and PrintObjectTransformedInputStream to obtain different transformations when reading spooled file data
- Example: Copying a spooled file shows how to copy a spooled file to the same queue that contains the file you want to copy.
- Example: Listing spooled files asynchronously (using listeners) shows how to asynchronously list all spooled files on a system and how to use the PrintObjectListListener interface to get feedback as the list is being built
- Example: Listing spooled files asynchronously (without using listeners) shows how to asynchronously list all spooled files on a system *without* using the PrintObjectListListener interface
- Example: Listing spooled files synchronously shows how to synchronously list all spooled files on a system

Listing Print objects:

You can use the PrintObjectList class and its subclasses to work with lists of print objects. Each subclass has methods that allow filtering of the list based on what makes sense for that particular type of print object. For example, SpooledFileList allows you to filter a list of spooled files based on the user who created the spooled files, the output queue that the spooled files are on, the form type, or user data of the spooled files. Only those spooled files that match the filter criteria are listed. If no filters are set, a default for each of the filters are used.

To actually retrieve the list of print objects from the server, the openSynchronously() or openAsynchronously() methods are used. The openSynchronously() method does not return until all objects in the list have been retrieved from the server. The openAsynchronously() method returns immediately, and the caller can do other things in the foreground while waiting for the list to build. The asynchronously opened list also allows the caller to start displaying the objects to the user as the objects come back. Because the user can see the objects as they come back, the response time may seem faster to the user. In fact, the response time may actually take longer overall due to the extra processing being done on each object in the list.

If the list is opened asynchronously, the caller may get feedback on the building of the list. Methods, such as isCompleted() and size(), indicate whether the list has finished being built or return the current size of the list. Other methods, waitForListToComplete() and waitForItem(), allow the caller to wait for the list to complete or for a particular item. In addition to calling these PrintObjectList methods, the caller may register with the list as a listener. In this situation, the caller is notified of events that happen to the list. To register or unregister for the events, the caller uses PrintObjectListListener(), and then calls addPrintObjectListListener() to register or removePrintObjectListListener() to unregister. The following table shows the events that are delivered from a PrintObjectList.

PrintObjectList event	When event is delivered
listClosed	When the list is closed.
listCompleted	When the list completes.
listErrorOccurred	If any exception is thrown while the list is being retrieved.
listOpened	When the list is opened.
listObjectAdded	When an object is added to the list.

After the list has been opened and the objects in the list processed, close the list using the close() method. This frees up any resources allocated to the garbage collector during the open. After a list has been closed, its filters can be modified, and the list can be opened again.

When print objects are listed, attributes about each print object listed are sent from the server and stored with the print object. These attributes can be updated using the update() method in the PrintObject class. Which attributes are sent back from the server depends on the type of print object being listed. A default list of attributes for each type of print object that can be overridden by using the setAttributesToRetrieve() method in PrintObjectList exists. See the Retrieving PrintObject attributes section for a list of the attributes each type of print object supports.

Listing AFP Resources is allowed only on Version 3 Release 7 and later releases of i5/OS. Opening an AFPResourceList to an system older than V3R7 generates a RequestNotSupportedException exception.

Examples

The following examples show different ways to list spooled files.

"Example: Listing spooled files asynchronously (using listeners)" on page 492 shows how to asynchronously list all spooled files on a system and how to use the PrintObjectListListener interface to get feedback as the list is being built

"Example: Listing spooled files asynchronously (without using listeners)" on page 495 shows how to asynchronously list all spooled files on a system *without* using the PrintObjectListListener interface

"Example: Listing spooled files synchronously" on page 497 shows how to synchronously list all spooled files on a system

Working with Print objects:

PrintObject is an abstract class. (An abstract class does not allow you to create an instance of the class. Instead, you must create an instance of one of its subclasses.) Create objects of the subclasses in any of the following ways:

- If you know the system and the identifying attributes of the object, construct the object explicitly by calling its public constructor.
- You can use a PrintObjectList subclass to build a list of the objects and then get at the individual objects through the list.
- An object may be created and returned to you as a result of a method or set methods being called. For example, the static method start() in the WriterJob class returns a WriterJob object.

Use the base class, PrintObject, and its subclasses to work with server print objects:

- OutputQueue
- Printer
- PrinterFile
- SpooledFile
- WriterJob

Retrieving PrintObject attributes:

You can retrieve print object attributes by using the attribute ID and one of these methods from the base PrintObject class:

- Use getIntegerAttribute(int attributeID) to retrieve an integer type attribute.
- Use getFloatAttribute(int attributeID) to retrieve a floating point type attribute.

- Use getStringAttribute(int attributeID) to retrieve a string type attribute.

The attributeID parameter is an integer that identifies which attribute to retrieve. All of the IDs are defined as public constants in the base PrintObject class. The PrintAttributes file contains an entry of each attribute ID. The entry includes a description of the attribute and its type (integer, floating point, or string). For a list of which attributes may be retrieved using these methods, select the following links:

- AFPResourceAttrs for AFP Resources
- OutputQueueAttrs for output queues
- PrinterAttrs for printers
- PrinterFileAttrs for printer files
- SpooledFileAttrs for spooled files
- WriterJobAttrs for writer jobs

To achieve acceptable performance, these attributes are copied to the client. These attributes are copied either when the objects are listed, or the first time they are needed if the object was created implicitly. This keeps the object from going to the host every time the application needs to retrieve an attribute. This also makes it possible for the Java print object instance to contain out-of-date information about the object on the server. The user of the object can refresh all of the attributes by calling the update() method on the object. In addition, if the application calls any methods on the object that would cause the object's attributes to change, the attributes are automatically updated. For example, if an output queue has a status attribute of RELEASED (getStringAttribute(ATTR_OUTQSTS); returns a string of ″RELEASED″), and the hold() method is called on the output queue, getting the status attribute after that returns HELD.

setAttributes method

You can use the setAttributes method to change the attributes of spooled files and printer file objects. Select the following links for a list or which attributes may be set:

- PrinterFileAttrs file for printer files
- SpooledFileAttrs for spooled files

The setAttributes method takes a PrintParameterList parameter, which is a class that is used to hold a collection of attributes IDs and their values. The list starts out empty, and the caller can add attributes to the list by using the various setParameter() methods on it.

PrintParameterList class

You can use the PrintParameterList class to pass a group of attributes to a method that takes any of a number of attributes as parameters. For example, you can send a spooled file using TCP (LPR) by using the SpooledFile method, sendTCP(). The PrintParameterList object contains the required parameters for the send command, such as the remote system and queue, plus any optional parameters desired, such as whether to delete the spooled file after it is sent. In these cases, the method documentation gives a list of required and optional attributes. The PrintParameterList setParameter() method does not check which attributes you are setting and the values that you set them to. The PrintParameterList setParameter() method simply contains the values to pass along to the method. In general, extra attributes in the PrintParameterList are ignored, and illegal values on the attributes that are used are diagnosed on the server.

AFP Resource Attributes:

Retrieve Attributes

The following attributes may be retrieved for an AFP resource using the appropriate getIntegerAttribute(), getStringAttribute(), or getFloatAttribute() method :

- ATTR_AFP_RESOURCE - AFP resource Integrated File System Path

- ATTR_OBJEXTATTR - Object Extended Attribute
- ATTR_DESCRIPTION - Text Description
- ATTR_DATE - Date File Opened
- ATTR_TIME - Time File Opened
- ATTR_NUMBYTES - Number of bytes to read/write

Set Attributes

Attributes are not allowed to be set for an AFP resource.

Output queue attributes:

Retrieve attributes

The following attributes may be retrieved for an output queue using the appropriate getIntegerAttribute(), getStringAttribute(), or getFloatAttribute() method :

- ATTR_AUTHCHCK - Authority to Check
- ATTR_DATA_QUEUE - Data Queue Integrated File System Name
- ATTR_DISPLAYANY - Display any File
- ATTR_JOBSEPRATR - Job Separators
- ATTR_NUMFILES - Number of Files
- ATTR_NUMWRITERS - Number of Writers Started to Queue
- ATTR_OPCNTRL - Operator Controlled
- ATTR_ORDER - Order of Files On Queue
- ATTR_OUTPUT_QUEUE - Output Queue Integrated File System Name
- ATTR_OUTQSTS - Output Queue Status
- ATTR_PRINTER - Printer
- ATTR_SEPPAGE - Separator page
- ATTR_DESCRIPTION - Text Description
- ATTR_USRDEFOPT - User defined option(s)
- ATTR_USER_DEFINED_OBJECT - User defined object Integrated File System Name
- ATTR_USER_TRANSFORM_PROG - User transform program Integrated File System Name
- ATTR_USER_DRIVER_PROG - User driver program Integrated File System Name
- ATTR_WTRJOBNAME - Writer Job Name
- ATTR_WTRJOBNUM - Writer Job Number
- ATTR_WTRJOBSTS - Writer Job Status
- ATTR_WTRJOBUSER - Writer Job User Name

Set attributes

Attributes are not allowed to be set for an output queue.

Printer Attributes:

Retrieve Attributes

The following attributes may be retrieved for a printer using the appropriate getIntegerAttribute(), getStringAttribute(), or getFloatAttribute() method :

- ATTR_AFP - Advanced Function Printing

- ATTR_ALIGNFORMS - Align Forms
- ATTR_ALWDRTPRT - Allow Direct Print
- ATTR_BTWNCPYSTS - Between copies status
- ATTR_BTWNFILESTS - Between files status
- ATTR_CODEPAGE - Code Page
- ATTR_CHANGES - Changes
- ATTR_DEVCLASS - Device Class
- ATTR_DEVMODEL - Device Model
- ATTR_DEVTYPE - Device Type
- ATTR_DEVSTATUS - Device Status
- ATTR_DRWRSEP - Drawer for Separators
- ATTR_ENDPNDSTS - End pending status
- ATTR_FILESEP - File Separators
- ATTR_FONTID - Font Identifier
- ATTR_FORM_DEFINITION - Form Definition Integrated File System Name
- ATTR_FORMTYPE - Form Type
- ATTR_FORMTYPEMSG - Form Type Message
- ATTR_FORMFEED - Form Feed
- ATTR_CHAR_ID - Graphic Character Set
- ATTR_HELDSTS - Held status
- ATTR_HOLDPNDSTS - Hold pending status
- ATTR_JOBUSER - Job User
- ATTR_MFGTYPE - Manufacturer Type and Model
- ATTR_MESSAGE_QUEUE - Message Queue Integrated File System Name
- ATTR_ONJOBQSTS - On job queue status
- ATTR_OUTPUT_QUEUE - Output Queue Integrated File System Name
- ATTR_OVERALLSTS - Overall Status
- ATTR_POINTSIZE - Point Size
- ATTR_PRINTER - Printer
- ATTR_PRTDEVTYPE - Printer Device Type
- ATTR_PUBINF_COLOR_SUP - Publishing Info Color Supported
- ATTR_PUBINF_PPM_COLOR - Publishing Info Pages per Minute (Color)
- ATTR_PUBINF_PPM - Publishing Info Pages per Minute (Monochrome)
- ATTR_PUBINF_DUPLEX_SUP - Publishing Info Duplex Support
- ATTR_PUBINF_LOCATION - Publishing Info Location
- ATTR_RMTLOCNAME - Remote Location Name
- ATTR_SPOOLFILE - Spooled File Name
- ATTR_SPLFNUM - Spooled File Number
- ATTR_STARTEDBY - Started By User
- ATTR_DESCRIPTION - Text Description
- ATTR_USERDATA - User data
- ATTR_USRDEFOPT - User defined option(s)
- ATTR_USER_DEFINED_OBJECT - User defined object Integrated File System Name
- ATTR_USER_TRANSFORM_PROG - User transform program Integrated File System Name
- ATTR_USER_DRIVER_PROG - User driver program Integrated File System Name

- ATTR_SCS2ASCII - Transform SCS to ASCII
- ATTR_WTNGDATASTS - Waiting for data status
- ATTR_WTNGDEVSTS - Waiting for device status
- ATTR_WTNGMSGSTS - Waiting for message status
- ATTR_WTRAUTOEND - When to Automatically End Writer
- ATTR_WTRJOBNAME - Writer Job Name
- ATTR_WTRJOBSTS - Writer Job Status
- ATTR_WTRSTRTD - Writer started
- ATTR_WRTNGSTS - Writing status

Set Attributes

Attributes are not allowed to be set for a printer.

Printer file attributes:

Retrieve attributes

The following attributes may be retrieved for a printer file using the appropriate getIntegerAttribute(), getStringAttribute(), or getFloatAttribute() method :

- ATTR_ALIGN - Align Page
- ATTR_BKMGN_ACR - Back Margin Offset Across
- ATTR_BKMGN_DWN - Back Margin Offset Down
- ATTR_BACK_OVERLAY - Back Overlay Integrated File System Name
- ATTR_BKOVL_DWN - Back Overlay Offset Down
- ATTR_BKOVL_ACR - Back Overlay offset across
- ATTR_CPI - Characters per Inch
- ATTR_CODEDFNTLIB - Coded Font Library Name
- ATTR_CODEPAGE - Code Page
- ATTR_CODEDFNT - Code Font Name
- ATTR_CONTROLCHAR - Control Character
- ATTR_CONVERT_LINEDATA - Convert Line Data
- ATTR_COPIES - Copies
- ATTR_CORNER_STAPLE - Corner staple
- ATTR_DBCSDATA - User Specified DBCS Data
- ATTR_DBCSEXTENSN - DBCS Extension Characters
- ATTR_DBCSROTATE - DBCS Character Rotation
- ATTR_DBCSCPI - DBCS Characters per Inch
- ATTR_DBCSSISO - DBCS SO/SI Spacing
- ATTR_DFR_WRITE - Defer Write
- ATTR_PAGRTT - Degree of Page Rotation
- ATTR_EDGESTITCH_NUMSTAPLES - Edge Stitch Number of Staples
- ATTR_EDGESTITCH_REF - Edge Stitch Reference
- ATTR_EDGESTITCH_REFOFF - Edge Stitch Reference
- ATTR_ENDPAGE - Ending Page
- ATTR_FILESEP - File Separators
- ATTR_FOLDREC - Fold Records

- ATTR_FONTID - Font Identifier
- ATTR_FORM_DEFINITION - Form Definition Integrated File System Name
- ATTR_FORMFEED - Form Feed
- ATTR_FORMTYPE - Form Type
- ATTR_FTMGN_ACR - Front Margin Offset Across
- ATTR_FTMGN_DWN - Front Margin Offset Down
- ATTR_FRONT_OVERLAY - Front overlay Integrated File System Name
- ATTR_FTOVL_ACR - Front Overlay Offset Across
- ATTR_FTOVL_DWN - Front Overlay Offset Down
- ATTR_CHAR_ID - Graphic Character Set
- ATTR_JUSTIFY - Hardware Justification
- ATTR_HOLD - Hold Spool File
- ATTR_LPI - Lines Per Inch
- ATTR_MAXRCDS - Maximum Spooled Output Records
- ATTR_OUTPTY - Output Priority
- ATTR_OUTPUT_QUEUE - Output Queue Integrated File System Name
- ATTR_OVERFLOW - Overflow Line Number
- ATTR_PAGE_DEFINITION - Page Definition Integrated File System
- ATTR_PAGELEN - Length of Page
- ATTR_MEASMETHOD - Measurement Method
- ATTR_PAGEWIDTH - Width of Page
- ATTR_MULTIUP - Pages Per Side
- ATTR_POINTSIZE - Point Size
- ATTR_FIDELITY - Print Fidelity
- ATTR_DUPLEX - Print on Both Sides
- ATTR_PRTQUALITY - Print Quality
- ATTR_PRTTEXT - Print Text
- ATTR_PRINTER - Printer
- ATTR_PRTDEVTYPE - Printer Device Type
- ATTR_RPLUNPRT - Replace Unprintable Characters
- ATTR_RPLCHAR - Replacement Character
- ATTR_SADDLESTITCH_NUMSTAPLES - Saddle Stitch Number of Staples
- ATTR_SADDLESTITCH_REF - Saddle Stitch Reference
- ATTR_SAVE -Save Spooled File
- ATTR_SRCDRWR - Source Drawer
- ATTR_SPOOL - Spool the Data
- ATTR_SCHEDULE - Spooled Output Schedule
- ATTR_STARTPAGE - Starting Page
- ATTR_DESCRIPTION - Text Description
- ATTR_UNITOFMEAS - Unit of Measure
- ATTR_USERDATA - User Data
- ATTR_USRDEFDATA - User defined data
- ATTR_USRDEFOPT - User defined option(s)
- ATTR_USER_DEFINED_OBJECT - User defined object Integrated File System Name

Set attributes

The following attributes may be set for a printer file using the setAttributes() method:

- ATTR_ALIGN - Align Page
- ATTR_BKMGN_ACR - Back Margin Offset Across
- ATTR_BKMGN_DWN - Back Margin Offset Down
- ATTR_BACK_OVERLAY - Back Overlay Integrated File System Name
- ATTR_BKOVL_DWN - Back Overlay Offset Down
- ATTR_BKOVL_ACR - Back Overlay offset across
- ATTR_CPI - Characters per Inch
- ATTR_CODEDFNTLIB - Coded Font Library Name
- ATTR_CODEPAGE - Code Page
- ATTR_CODEDFNT - Code Font Name
- ATTR_CONTROLCHAR - Control Character
- ATTR_CONVERT_LINEDATA - Convert Line Data
- ATTR_COPIES - Copies
- ATTR_CORNER_STAPLE - Corner staple
- ATTR_DBCSDATA - User Specified DBCS Data
- ATTR_DBCSEXTENSN - DBCS Extension Characters
- ATTR_DBCSROTATE - DBCS Character Rotation
- ATTR_DBCSCPI - DBCS Characters per Inch
- ATTR_DBCSSISO - DBCS SO/SI Spacing
- ATTR_DFR_WRITE - Defer Write
- ATTR_PAGRTT - Degree of Page Rotation
- ATTR_EDGESTITCH_NUMSTAPLES - Edge Stitch Number of Staples
- ATTR_EDGESTITCH_REF - Edge Stitch Reference
- ATTR_EDGESTITCH_REFOFF - Edge Stitch Reference
- ATTR_ENDPAGE - Ending Page
- ATTR_FILESEP - File Separators
- ATTR_FOLDREC - Fold Records
- ATTR_FONTID - Font Identifier
- ATTR_FORM_DEFINITION - Form Definition Integrated File System Name
- ATTR_FORMFEED - Form Feed
- ATTR_FORMTYPE - Form Type
- ATTR_FTMGN_ACR - Front Margin Offset Across
- ATTR_FTMGN_DWN - Front Margin Offset Down
- ATTR_FRONT_OVERLAY - Front overlay Integrated File System Name
- ATTR_FTOVL_ACR - Front Overlay Offset Across
- ATTR_FTOVL_DWN - Front Overlay Offset Down
- ATTR_CHAR_ID - Graphic Character Set
- ATTR_JUSTIFY - Hardware Justification
- ATTR_HOLD - Hold Spool File
- ATTR_LPI - Lines Per Inch
- ATTR_MAXRCDS - Maximum Spooled Output Records
- ATTR_OUTPTY - Output Priority

- ATTR_OUTPUT_QUEUE - Output Queue Integrated File System Name
- ATTR_OVERFLOW - Overflow Line Number
- ATTR_PAGE_DEFINITION - Page Definition Integrated File System
- ATTR_PAGELEN - Length of Page
- ATTR_MEASMETHOD - Measurement Method
- ATTR_PAGEWIDTH - Width of Page
- ATTR_MULTIUP - Pages Per Side
- ATTR_POINTSIZE - Point Size
- ATTR_FIDELITY - Print Fidelity
- ATTR_DUPLEX - Print on Both Sides
- ATTR_PRTQUALITY - Print Quality
- ATTR_PRTTEXT - Print Text
- ATTR_PRINTER - Printer
- ATTR_PRTDEVTYPE - Printer Device Type
- ATTR_RPLUNPRT - Replace Unprintable Characters
- ATTR_RPLCHAR - Replacement Character
- ATTR_SADDLESTITCH_NUMSTAPLES - Saddle Stitch Number of Staples
- ATTR_SADDLESTITCH_REF - Saddle Stitch Reference
- ATTR_SAVE -Save Spooled File
- ATTR_SRCDRWR - Source Drawer
- ATTR_SPOOL - Spool the Data
- ATTR_SCHEDULE - Spooled Output Schedule
- ATTR_STARTPAGE - Starting Page
- ATTR_DESCRIPTION - Text Description
- ATTR_UNITOFMEAS - Unit of Measure
- ATTR_USERDATA - User Data
- ATTR_USRDEFDATA - User defined data
- ATTR_USRDEFOPT - User defined option(s)
- ATTR_USER_DEFINED_OBJECT - User defined object Integrated File System Name

Spooled file attributes:

Retrieve attributes

The following attributes may be retrieved for a spooled file using the appropriate getIntegerAttribute(), getStringAttribute(), or getFloatAttribute() method:

- ATTR_AFP - Advanced Function Printing
- ATTR_ALIGN - Align Page
- ATTR_BKMGN_ACR - Back Overlay offset across
- ATTR_BKMGN_DWN - Back Overlay Offset Down
- ATTR_BACK_OVERLAY - Back Overlay Integrated File System Name
- ATTR_BKOVL_DWN - Back Overlay Offset Down
- ATTR_BKOVL_ACR - Back Overlay offset across
- ATTR_CPI - Characters per Inch
- ATTR_CODEDFNTLIB - Coded Font Library Name
- ATTR_CODEDFNT - Code Font Name

- ATTR_CODEPAGE - Code Page
- ATTR_CONTROLCHAR - Control Character
- ATTR_COPIES - Copies
- ATTR_COPIESLEFT - Copies left to Produce
- ATTR_CORNER_STAPLE - Corner staple
- ATTR_CURPAGE - Current Page
- ATTR_DATE - Date Object Created
- ATTR_DATE_WTR_BEGAN_FILE - Date Writer Began Processing Spooled File
- ATTR_DATE_WTR_CMPL_FILE - Date Writer Completed Processing Spooled File
- ATTR_DBCSDATA - User Specified DBCS Data
- ATTR_DBCSEXTENSN - DBCS Extension Characters
- ATTR_DBCSROTATE - DBCS Character Rotation
- ATTR_DBCSCPI - DBCS Characters per Inch
- ATTR_DBCSSISO - DBCS SO/SI Spacing
- ATTR_PAGRTT - Degree of Page Rotation
- ATTR_EDGESTITCH_NUMSTAPLES - Edge Stitch Number of Staples
- ATTR_EDGESTITCH_REF - Edge Stitch Reference
- ATTR_EDGESTITCH_REFOFF - Edge Stitch Reference Offset
- ATTR_ENDPAGE - Ending Page
- ATTR_FILESEP - File Separators
- ATTR_FOLDREC - Fold Records
- ATTR_FONTID - Font Identifier
- ATTR_FORM_DEFINITION - Form definition Integrated File System Name
- ATTR_FORMFEED - Form Feed
- ATTR_FORMTYPE - Form Type
- ATTR_FTMGN_ACR - Front Margin Offset Across
- ATTR_FTMGN_DWN - Front Margin Offset Down
- ATTR_FRONTSIDE_OVERLAY - Front overlay Integrated File System Name
- ATTR_FTOVL_ACR - Front Overlay Offset Across
- ATTR_FTOVL_DWN - Front Overlay Offset Down
- ATTR_CHAR_ID - Graphic Character Set
- ATTR_JUSTIFY - Hardware Justification
- ATTR_HOLD - Hold Spool File
- ATTR_IPP_ATTR_CHARSET - IPP Attributes-charset
- ATTR_IPP_JOB_ID - IPP Job ID
- ATTR_IPP_JOB_NAME - IPP Job Name
- ATTR_IPP_JOB_NAME_NL - IPP Job Name NL
- ATTR_IPP_JOB_ORIGUSER - IPP Job Originating User
- ATTR_IPP_JOB_ORIGUSER_NL - IPP Job Originating User NL
- ATTR_IPP_PRINTER_NAME - IPP Printer Name
- ATTR_JOBNAME - Job Name
- ATTR_JOBNUMBER - Job Number
- ATTR_JOBUSER - Job User
- ATTR_JOB_SYSTEM - Job System
- ATTR_LASTPAGE - Last Page Printed

- ATTR_LINESPACING - Line Spacing
- ATTR_LPI - Lines Per Inch
- ATTR_MAXRCDS - Maximum Spooled Output Records
- ATTR_PAGELEN - Length of Page
- ATTR_PAGEWIDTH - Width of Page
- ATTR_MEASMETHOD - Measurement Method
- ATTR_NETWORK - Network Identifier
- ATTR_NUMBYTES - Number of bytes to read/write
- ATTR_OUTPUTBIN - Output Bin
- ATTR_OUTPTY - Output Priority
- ATTR_OUTPUT_QUEUE - Output Queue Integrated File System Name
- ATTR_OVERFLOW - Overflow Line Number
- ATTR_MULTIUP - Pages Per Side
- ATTR_POINTSIZE - Point Size
- ATTR_FIDELITY - Print Fidelity
- ATTR_DUPLEX - Print on Both Sides
- ATTR_PRTQUALITY - Print Quality
- ATTR_PRTTEXT - Print Text
- ATTR_PRINTER - Printer
- ATTR_PRTASSIGNED - Printer Assigned
- ATTR_PRTDEVTYPE - Printer Device Type
- ATTR_PRINTER_FILE - Printer File Integrated File System Name
- ATTR_RECLENGTH - Record Length
- ATTR_REDUCE - Reduce Output
- ATTR_RPLUNPRT - Replace Unprintable Characters
- ATTR_RPLCHAR - Replacement Character
- ATTR_RESTART - Restart Printing
- ATTR_SADDLESTITCH_NUMSTAPLES - Saddle Stitch Number of Staples
- ATTR_SADDLESTITCH_REF - Saddle Stitch Reference
- ATTR_SAVE - Save Spooled File
- ATTR_SRCDRWR - Source Drawer
- ATTR_SPOOLFILE - Spooled File Name
- ATTR_SPLFNUM - Spooled File Number
- ATTR_SPLFSTATUS - Spooled File Status
- ATTR_SCHEDULE - Spooled Output Schedule
- ATTR_STARTPAGE - Starting Page
- ATTR_SYSTEM - System Where Created
- ATTR_TIME - Time Object Created
- ATTR_TIME_WTR_BEGAN_FILE - Time Writer Began Processing Spooled File
- ATTR_TIME_WTR_CMPL_FILE - Time Writer Completed Processing Spooled File
- ATTR_PAGES - Total Pages
- ATTR_UNITOFMEAS - Unit of Measure
- ATTR_USERCMT - User Comment
- ATTR_USERDATA - User Data
- ATTR_USRDEFDATA - User Defined Data

- ATTR_USRDEFFILE - User Defined File
- ATTR_USRDEFOPT - User Defined Option(s)
- ATTR_USER_DEFINED_OBJECT - User Defined Object Integrated File System Name

Set attributes

The following attributes may be set for a spooled file using the setAttributes() method:
- ATTR_ALIGN - Align Page
- ATTR_BACK_OVERLAY - Back Overlay Integrated File System Name
- ATTR_BKOVL_DWN - Back Overlay Offset Down
- ATTR_BKOVL_ACR - Back Overlay offset across
- ATTR_COPIES - Copies
- ATTR_ENDPAGE - Ending Page
- ATTR_FILESEP - File Separators
- ATTR_FORM_DEFINITION - Form definition Integrated File System Name
- ATTR_FORMFEED - Form Feed
- ATTR_FORMTYPE - Form Type
- ATTR_FRONTSIDE_OVERLAY - Front overlay Integrated File System Name
- ATTR_FTOVL_ACR - Front Overlay Offset Across
- ATTR_FTOVL_DWN - Front Overlay Offset Down
- ATTR_OUTPTY - Output Priority
- ATTR_OUTPUT_QUEUE - Output Queue Integrated File System Name
- ATTR_MULTIUP - Pages Per Side
- ATTR_FIDELITY - Print Fidelity
- ATTR_DUPLEX - Print on Both Sides
- ATTR_PRTQUALITY - Print Quality
- ATTR_PRTSEQUENCE - Print Sequence
- ATTR_PRINTER - Printer
- ATTR_RESTART - Restart Printing
- ATTR_SAVE - Save Spooled File
- ATTR_SCHEDULE - Spooled Output Schedule
- ATTR_STARTPAGE - Starting Page
- ATTR_USERDATA - User Data
- ATTR_USRDEFOPT - User defined option(s)
- ATTR_USER_DEFINED_OBJECT - User defined object Integrated File System Name

Writer Job Attributes:

Retrieve Attributes

The following attributes may be retrieved for a writer job using the appropriate getIntegerAttribute(), getStringAttribute(), or getFloatAttribute() method :
- ATTR_WTRJOBNAME - Writer Job Name
- ATTR_WTRJOBNUM - Writer Job Number
- ATTR_WTRJOBSTS - Writer Job Status
- ATTR_WTRJOBUSER - Writer Job User Name

Set Attributes

Attributes are not allowed to be set for a writer job.

Print Object Attributes:

Table of Contents

- Destination Option
- Destination Type
- Device Class
- Device Model
- Device Status
- Device Type
- Display any File
- Drawer for Separators
- Edge Stitch Number of Staples
- Edge Stitch Reference
- Edge Stitch Reference Offset
- End Pending Status
- Ending Page
- Envelope Source
- File Separators
- Fold Records
- Font Identifier
- Form Definition
- Form Feed
- Form Type
- Form Type Message Option
- Front Margin Offset Across
- Front Margin Offset Down
- Front Overlay
- Front Overlay Offset Across
- Front Overlay Offset Down
- Graphic Character Set
- Hardware Justification
- Held Status
- Hold Spool File
- Hold Pending Status
- Image Configuration
- Initialize the writer
- Internet Address
- IPP Attributes-charset
- IPP Job ID
- IPP Job Name
- IPP Job Name NL
- IPP Job Originating User Name
- IPP Job Originating User Name NL
- IPP IPP Printer Name
- Job Name
- Job Number
- Job Separators
- Job System

- Job User
- Last Page Printed
- Length of Page
- Library Name
- Lines Per Inch
- Line Spacing
- Manufacturer Type and Model
- Max Jobs per Client List
- Maximum Spooled Output Records
- Measurement Method
- Message Help
- Message ID
- Message Queue
- Message Reply
- Message Text
- Message Type
- Message Severity
- Multi_Item Reply Capability
- Network Identifier
- Network Print Server Object Attributes
- Number of Bytes in Spooled File
- Number of Bytes to Read/Write
- Number of Files
- Number of Writers Started to Queue
- Object Extended Attribute
- On Job Queue Status
- Open time commands
- Operator Controlled
- Order of Files On Queue
- Output Bin
- Output Priority
- Output Queue
- Output Queue Status
- Overall Status
- Overflow Line Number
- Page At A Time
- Page Count Estimated
- Page Definition
- Page Number
- Pages Per Side
- Paper Source 1
- Paper Source 2
- Pel Density
- Point Size
- Print Fidelity

- Print on Both Sides
- Print Quality
- Print Sequence
- Print Text
- Printer
- Printer Assigned
- Printer Device Type
- Printer File
- Printer Queue
- Publishing Info Color Supported
- Publishing Info Pages per Minute (Color)
- Publishing Info Pages per Minute (Monochrome)
- Publishing Info Duplex Support
- Publishing Info Location
- Remote Location Name
- Record Length
- Reduce Output
- Remote System
- Replace Unprintable Characters
- Replacement Character
- Restart Printing
- Saddle Stitch Number of Staples
- Saddle Stitch Reference
- Save Spooled File
- Seek Offset
- Seek Origin
- Send Priority
- Separator page
- Source Drawer
- Spool SCS
- Spool the Data
- Spooled File Creation Authentication Method
- Spooled File Creation Security Method
- Spooled File Name
- Spooled File Number
- Spooled File Status
- Spooled Output Schedule
- Started By User
- Starting Page
- System Where Created
- Text Description
- Time File Opened
- Time Spooled File Create Job End
- Time Writer Began Processing Spooled File
- Time Writer Completed Processing Spooled File

- Total Pages
- Transform SCS to ASCII
- Unit of Measure
- User Comment
- User Data
- User Defined Data
- User Defined File
- User Defined Object
- User Defined Option(s)
- User Driver Program Data
- User Driver Program
- User ID
- User ID Address
- User Transform Program
- Viewing Fidelity
- VM/MVS Class
- Waiting for Data Status
- Waiting for Device Status
- Waiting for Message Status
- When to Automatically End Writer
- When to End Writer
- When to Hold File
- Width of Page
- Workstation Customizing Object
- Writer Job Name
- Writer Job Number
- Writer Job Status
- Writer Job User Name
- Writer Started
- Writer Starting Page
- Writing Status
- NPS CCSID
- NPS Level

Advanced Function Printing

ID ATTR_AFP

Type String

Description
Indicates whether this spooled file uses AFP resources external to the spooled file. Valid values are *YES and *NO.

AFP Resource

ID ATTR_AFP_RESOURCE

Type String

Description
The Integrated File System path of the external AFP (Advanced Function Print) resource. The format of the Integrated File System path is ″/QSYS.LIB/library.LIB/resource.type″ where *library* is the library that contains the resource, *resource* is the name of the resource and *type* is the resource type. Valid values for *type* include ″FNTRSC″, ″FORMDF″, ″OVL″, ″PAGSEG″, and ″PAGDFN″.

Align Forms

ID ATTR_ALIGNFORMS

Type String

Description
The time at which a forms alignment message will be sent. Valid values are *WTR, *FILE, *FIRST.

Align Page

ID ATTR_ALIGN

Type String

Description
Indicates whether a forms alignment message is sent before printing this spooled file. Valid values are *YES, *NO.

Allow Direct Print

ID ATTR_ALWDRTPRT

Type String

Description
Indicates whether the printer writer allows the printer to be allocated to a job that prints directly to a printer. Valid values are *YES, *NO.

Authority

ID ATTR_AUT

Type String

Description
Specifies the authority that is given to users who do not have specific authority to the output queue. Valid values are *USE, *ALL, *CHANGE, *EXCLUDE, *LIBCRTAUT.

Authority to Check

ID ATTR_AUTCHK

Type String

Description
Indicates what type of authorities to the output queue allow the user to control all the files on the output queue. Valid values are *OWNER, *DTAAUT.

Automatically End Writer

ID ATTR_AUTOEND

Type String

Description
Specifies if the writer must be automatically ended. Valid values are *NO, *YES.

Auxiliary Storage

ID ATTR_AUX_POOL

Type Integer

Description

Specifies the number of the auxiliary storage pool (ASP) that the spooled file is stored on. The possible values are:

- 1: System ASP
- 2-32: One of the user ASPs

Back Margin Offset Across

ID ATTR_BACKMGN_ACR

Type Float

Description

For the back side of a piece of paper, it specifies, how far in from the left side of the page printing starts. The special value *FRONTMGN will be encoded as -1.

Back Margin Offset Down

ID ATTR_BACKMGN_DWN

Type Float

Description

For the back side of a piece of paper, it specifies, how far down from the top of the page printing starts. The special value *FRONTMGN will be encoded as -1.

Back Overlay

ID ATTR_BACK_OVERLAY

Type String

Description

The Integrated File System path of the back overlay or a special value. If the value is an Integrated File System path it will have the format ″/QSYS.LIB/library.LIB/overlay.OVL″ where *library* is the library of the resource and *overlay* is the name of the overlay. Valid special values include *FRONTOVL.

Back Overlay offset across

ID ATTR_BKOVL_ACR

Type Float

Description

The offset across from the point of origin where the overlay is printed.

Back Overlay Offset Down

ID ATTR_BKOVL_DWN

Type Float

Description

The offset down from the point of origin where the overlay is printed.

Between Copies Status

ID ATTR_BTWNCPYSTS

Type String

Description
Whether the writer is between copies of a multiple copy spooled file. Returned values are *YES or *NO.

Between Files Status

ID ATTR_BTWNFILESTS

Type String

Description
Whether the writer is between files. Returned values are *YES or *NO.

Changes

ID ATTR_CHANGES

Type String

Description
The time at which pending changes take effect. Valid values are *NORDYF, *FILEEND, or blank which implies no changes pending to the writer.

Characters per Inch

ID ATTR_CPI

Type Float

Description
The number of characters per horizontal inch.

Code Page

ID ATTR_CODEPAGE

Type String

Description
The mapping of graphic characters to code points for this spooled file. If the graphic character set field contains a special value, this field may contain a zero (0).

Code Font Name

ID ATTR_CODEDFNT

Type String

Description
The name of the coded font. A coded font is an AFP resource that is composed of a character set and a code page. Special values include *FNTCHRSET.

Coded Font Library Name

ID ATTR_CODEDFNTLIB

Type String

Description
The name of the library that contains the coded font. This field may contain blanks if the coded font name field has a special value.

Control Character

ID ATTR_CONTROLCHAR

Type String

Description
Whether this file uses the American National Standards printer control character. The possible values are *NONE for no print control characters are passed in the data that is printed or *FCFC which means that the first character of every record is an American National Standards printer control character.

Convert Line Data

ID ATTR_CONVERT_LINEDATA

Type String

Description
Whether the line data is converted to AFPDS before it is written to spool. The possible values are *NO and *YES.

Copies

ID ATTR_COPIES

Type Integer

Description
The total number of copies to be produced for this spooled file.

Copies left to Produce

ID ATTR_COPIESLEFT

Type Integer

Description
The remaining number of copies to be produced for this spooled file.

Corner Staple

ID ATTR_CORNER_STAPLE

Type String

Description
The reference corner to be used for a corner staple. A staple is driven into the media at the reference corner. Valid values are *NONE, *DEVD, *BOTRIGHT, *TOPRIGHT, *TOPLEFT, and *BOTLEFT.

Current Page

ID ATTR_CURPAGE

Type Integer

Description
Current page that is being written by the writer job.

Data Format

ID ATTR_DATAFORMAT

Type String

Description
Data format. Valid values are *RCDDATA, *ALLDATA.

Data Queue

ID ATTR_DATA_QUEUE

Type String

Description
Specifies the Integrated File System path of the data queue that is associated with the output queue or "*NONE" if no data queue is associated with the the output queue. The format of the Integrated File System path is "/QSYS.LIB/library.LIB/dataqueue.DTAQ" where *library* is the library that contains the data queue and *dataqueue* is the name of the data queue.

Date File Opened

ID ATTR_DATE

Type String

Description
For spooled files this is the date the spooled file was opened. For AFP resources this is the date the object was last modified. The date is encoded in a character string with the following format, C YY MM DD.

Date Spooled File Job Create End

ID ATTR_DATE_END

Type String

Description
The date the job ended that created the spooled file on the system. If the Starting spooled file create date field is set to *ALL, then this field must be set to blanks. If a date has been specified for the Starting spooled file create date field, then this field must be set to a valid date. The date must be in the CYYMMDD format or be one of the following special values:

- *LAST: All spooled files with a create date and time equal to or later than the starting spooled file create date are to be returned.
- Date: All spooled files with a create date and time equal to or later than the starting spooled file create date and time and less than or equal to the ending spooled file create date and time are to be returned.

The date format CYYMMDD is defined as follows:

- C is the Century, where 0 indicates years 19xx and 1 indicates years 20xx
- YY is the Year
- MM is the Month
- DD is the Day

Date Writer Began Processing Spooled File

ID ATTR_DATE_WTR_BEGAN_FILE

Type String

Description
Indicates the date at which the writer began processing this spooled file. The date is encoded in a character string with the following format, C YY MM DD.

Date Writer Completed Processing Spooled File

ID ATTR_DATE_WTR_CMPL_FILE

Type String

Description
Indicates the date at which the writer began finished this spooled file. The date is encoded in a character string with the following format, C YY MM DD.

User Specified DBCS Data

ID ATTR_DBCSDATA

Type String

Description
Whether the spooled file contains double-byte character set (DBCS) data. Valid values are *NO and *YES.

DBCS Extension Characters

ID ATTR_DBCSEXTENSN

Type String

Description
Whether the system is to process the DBCS extension characters. Valid values are *NO and *YES.

DBCS Character Rotation

ID ATTR_DBCAROTATE

Type String

Description
Whether the DBCS characters are rotated 90 degrees counterclockwise before printing. Valid values are *NO and *YES.

DBCS Characters per Inch

ID ATTR_DBCSCPI

Type Integer

Description
The number of double-byte characters to be printed per inch. Valid values are -1, -2, 5, 6, and 10. The value *CPI is encoded as -1. The value *CONDENSED is encoded as -2.

DBCS SO/SI Spacing

ID ATTR_DBCSSISO

Type String

Description
Determines the presentation of shift-out and shift-in characters when printed. Valid values are *NO, *YES, and *RIGHT.

Defer Write

ID ATTR_DFR_WRITE

Type String

Description
Whether print data is held in system buffers before

Degree of Page Rotation

ID ATTR_PAGRTT

Type Integer

Description
The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. Valid values are -1, -2, -3, 0, 90, 180, 270. The value *AUTO is encoded as -1, the value *DEVD is encoded as -2, and the value *COR is encoded as -3.

Delete File After Sending

ID ATTR_DELETESPLF

Type String

Description
Delete the spooled file after sending? Valid values are *NO and *YES.

Destination Option

ID ATTR_DESTOPTION

Type String

Description
Destination option. A text string that allows the user to pass options to the receiving system.

Destination Type

ID ATTR_DESTINATION

Type String

Description
Destination type. Valid values are *OTHER, *AS400, *PSF2.

Device Class

ID ATTR_DEVCLASS

Type String

Description
The device class.

Device Model

ID ATTR_DEVMODEL

Type String

Description
The model number of the device.

Device Status

ID ATTR_DEVSTATUS

Type Integer

Description
The status of the printer device. Valid values are 0 (varied off), 10 (vary off pending), 20 (vary on pending), 30 (varied on), 40 (connect pending), 60 (active), 66 (active writer), 70 (held), 75 (powered off), 80 (recovery pending), 90 (recovery canceled), 100 (failed), 106 (failed writer), 110 (being serviced), 111 (damaged), 112 (locked), 113 (unknown).

Device Type

ID ATTR_DEVTYPE

Type String

Description
The device type.

Display any File

ID ATTR_DISPLAYANY

Type String

Description
Whether users who have authority to read this output queue can display the output data of any output file on this queue or only the data in their own files. Valid values are *YES, *NO, *OWNER.

Drawer for Separators

ID ATTR_DRWRSEP

Type Integer

Description
Identifies the drawer from which the job and file separator pages are to be taken. Valid values are -1, -2, 1, 2, 3. The value *FILE is encoded as -1, and the value *DEVD is encoded as -2.

Edge Stitch Number of Staples

ID ATTR_EDGESTITCH_NUMSTAPLES

Type Integer

Description
The number of staples that are applied along the finishing operation axis.

Edge Stitch Reference

ID ATTR_EDGESTITCH_REF

Type String

Description
Where one or more staples are driven into the media along the finishing operation axis. Valid values are *NONE, *DEVD, *BOTTOM, *RIGHT, *TOP, and *LEFT.

Edge Stitch Reference Offset

ID ATTR_EDGESTITCH_REFOFF

Type Float

Description
The offset of the edge stitch from the reference edge toward the center of the media.

End Pending Status

ID ATTR_ENDPNDSTS

Type String

Description

Whether an End Writer (ENDWTR) command has been issued for this writer. Possible values are *NO - no ENDWTR command was issued, *IMMED - the writer ends as soon as its output buffers are empty, *CTRLD - the writer ends after the current copy of the spooled file has printed, *PAGEEND - the writer ends at the end of the page.

Ending Page

ID ATTR_ENDPAGE

Type Integer

Description

The page number at which to end printing the spooled file. Valid values are 0 or the ending page number. The value *END is encoded as 0.

Envelope Source

ID ATTR_ENVLP_SOURCE

Type String

Description

The size of the envelope in the envelope source. If this field is not specified or the value is not valid, the special value of *MFRTYPMDL is used. Valid values are *NONE - there is no envelope source, *MFRTYPMDL - the envelope size suggested by the manufacturer type and model is used, *MONARCH (3.875 x 7.5 inches), *NUMBER9 (3.875 x 8.875 inches), *NUMBER10 (4.125 x 9.5 inches), *B5 (176mm x 250mm), *C5 (162mm x 229mm), *DL (110mm x 220mm).

File Separators

ID ATTR_FILESEP

Type Integer

Description

The number of file separator pages that are placed at the beginning of each copy of the spooled file. Valid values are -1, or the number of separators. The value *FILE is encoded as -1.

Fold Records

ID ATTR_FOLDREC

Type String

Description

Whether records that exceed the printer forms width are folded (wrapped) to the next line. Valid values are *YES, *NO.

Font Identifier

ID ATTR_FONTID

Type String

Description

The printer font that is used. Valid special values include *CPI and *DEVD.

Form Definition

ID ATTR_FORM_DEFINITION

Type String

Description
The Integrated File System path name of the form definition or a special value. If an Integrated File System path is specified the format is ″/QSYS.LIB/library.LIB/formdef.FORMDF″ where *library* is the library of the form definition and *formdef* is the name of the form definition. Valid special values include *NONE, *INLINE, *INLINED, and *DEVD.

Form Feed

ID ATTR_FORMFEED

Type String

Description
The manner in which forms feed to the printer. Valid values are *CONT, *CUT, *AUTOCUT, *DEVD.

Form Type

ID ATTR_FORMTYPE

Type String

Description
The type of form to be loaded in the printer to print this spooled file.

Form Type Message Option

ID ATTR_FORMTYPEMSG

Type String

Description
Message option for sending a message to the writer's message queue when the current form type is finished. Valid values are *MSG, *NOMSG, *INFOMSG, *INQMSG.

Front Margin Offset Across

ID ATTR_FTMGN_ACR

Type Float

Description
For the front side of a piece of paper, it specifies, how far in from the left side of the page printing starts. The special value *DEVD is encoded as -2.

Front Margin Offset Down

ID ATTR_FTMGN_DWN

Type Float

Description
For the front side of a piece of paper, it specifies, how far down from the top of the page printing starts. The special value *DEVD is encoded as -2.

Front Overlay

ID ATTR_FRONT_OVERLAY

Type String

Description
The Integrated File System path of the front overlay. The format of the Integrated File System path is "/QSYS.LIB/library.LIB/overlay.OVL" where *library* is the library of the resource and *overlay* is the name of the overlay. The string "*NONE" is used to indicate that no front overlay is specified.

Front Overlay Offset Across

ID ATTR_FTOVL_ACR

Type Float

Description
The offset across from the point of origin where the overlay is printed.

Front Overlay Offset Down

ID ATTR_FTOVL_DWN

Type Float

Description
The offset down from the point of origin where the overlay is printed.

Graphic Character Set

ID ATTR_CHAR_ID

Type String

Description
The set of graphic characters to be used when printing this file. Valid special values include *DEVD, *SYSVAL, and *JOBCCSID.

Hardware Justification

ID ATTR_JUSTIFY

Type Integer

Description
The percentage that the output is right justified. Valid values are 0, 50, 100.

Held Status

ID ATTR_HELDSTS

Type String

Description
Whether the writer is held. Valid values are *YES, *NO.

Hold Spool File

ID ATTR_HOLD

Type String

Description
Whether the spooled file is held. Valid values are *YES, *NO.

Hold Pending Status

ID ATTR_HOLDPNDSTS

Type String

Description
Whether a Hold Writer (HLDWTR) command has been issued for this writer. Possible values are *NO - no HLDWTR command was issued, *IMMED - the writer is held when its output buffers are empty, *CTRLD - writer held after the current copy of the spooled file has printed, *PAGEEND - writer held at the end of the page.

Image Configuration

ID ATTR_IMGCFG

Type String

Description
The transform services for a variety of image and print data-stream formats.

Initialize the writer

ID ATTR_WTRINIT

Type String

Description
The user can specify when to initialize the printer device. Valid values are *WTR, *FIRST, *ALL.

Internet Address

ID ATTR_INTERNETADDR

Type String

Description
The internet address of the receiving system.

IPP Attributes-charset

ID ATTR_IPP_ATTR_CHARSET

Type String

Description
Indicates the charset (coded character set and encoding method) of the IPP specified spooled file attributes.

IPP Job ID

ID ATTR_IPP_JOB_ID

Type Integer

Description
IPP Job ID relative to the IPP printer that created the job.

IPP Job Name

ID ATTR_IPP_ATR_CHARSET

Type String

Description
User friendly name of job.

IPP Job Name NL

ID ATTR_IPP_JOB_NAME_NL

Type String

Description
Natural language of job name.

IPP Job Originating User Name

ID ATTR_IPP_JOB_ORIGUSER

Type String

Description
Identifies the end user that submitted this IPP job.

IPP Job Originating User Name NL

ID ATTR_IPP_JOB_ORIGUSER_NL

Type String

Description
Identifies the natural language of job-originating user name.

IPP Printer Name

ID ATTR_IPP_PRINTER_NAME

Type String

Description
Identifies the IPP printer that created this job.

Job Name

ID ATTR_JOBNAME

Type String

Description
The name of the job that created the spooled file.

Job Number

ID ATTR_JOBNUMBER

Type String

Description
The number of the job that created the spooled file.

Job Separators

ID ATTR_JOBSEPRATR

Type Integer

Description
The number of job separators to be placed at the beginning of the output for each job having spooled files on this output queue. Valid values are -2, 0-9. The value *MSG is encoded as -2. Job separators are specified when the output queue is created.

Job System

ID ATTR_JOBSYSTEM

Type String

Description
The system job which created spooled file was running.

Job User

ID ATTR_JOBUSER

Type String

Description
The name of the user that created the spooled file.

Last Page Printed

ID ATTR_LASTPAGE

Type Integer

Description
The number of the last printed page is the file if printing ended before the job completed processing.

Length of Page

ID ATTR_PAGELEN

Type Float

Description
The length of a page. Units of measurement are specified in the measurement method attribute.

Library Name

ID ATTR_LIBRARY

Type String

Description
The name of the library.

Lines Per Inch

ID ATTR_LPI

Type Float

Description
The number of lines per vertical inch in the spooled file.

Line Spacing

ID ATTR_LINESPACING

Type String

Description
How a file's line data records are spaced when printed. The information is returned only for *LINE and *AFPDSLINE printer device types files. Valid values are *SINGLE, *DOUBLE, *TRIPLE, or *CTLCHAR.

Manufacturer Type and Model

ID ATTR_MFGTYPE

Type String

Description
Specifies the manufacturer, type, and model when transforming print data from SCS to ASCII.

Maximum Jobs per Client List

ID ATTR_MAX_JOBS_PER_CLIENT

Type Integer

Description
Supplied by the client to indicate the maximum printer queue size of limitation.

Maximum Spooled Output Records

ID ATTR_MAXRECORDS

Type Integer

Description
The maximum number of records allowed in this file at the time this file was opened. The value *NOMAX is encoded as 0.

Measurement Method

ID ATTR_MEASMETHOD

Type String

Description
The measurement method that is used for the length of page and width of page attributes. Valid values are *ROWCOL, *UOM.

Message Help

ID ATTR_MSGHELP

Type char(*)

Description
The message help, which is sometimes known as second-level text, can be returned by a ″retrieve message″ request. The system limits the length to 3000 characters (English version must be 30% less to allow for translation).

Message ID

ID ATTR_MESSAGEID

Type String

Description
The message ID.

Message Queue

ID ATTR_MESSAGE_QUEUE

Type String

Description
The Integrated File System path of the message queue that the writer uses for operational messages. The format of the Integrated File System path is ″/QSYS.LIB/library.LIB/messageque.MSGQ″ where *library* is the library that contains the message queue and *messageque* is the name of the message queue.

Message Reply

ID ATTR_MSGREPLY

Type String

Description

The message reply. Text string to be provided by the client which answers a message of type ″inquiry″. In the case of message retrieved, the attribute value is returned by the server and contains the default reply which the client can use. The system limits the length to 132 characters. Must be null-terminated due to variable length.

Message Text

ID ATTR_MSGTEXT

Type String

Description

The message text, that is sometimes known as first-level text, can be returned by a ″retrieve message″ request. The system limits the length to 132 characters.

Message Type

ID ATTR_MSGTYPE

Type String

Description

The message type, a 2-digit, EBCDIC encoding. Two types of messages indicate whether one can ″answer″ a ″retrieved″ message: ’04’ Informational messages convey information without asking for a reply (may require a corrective action instead), ’05’ Inquiry messages convey information and ask for a reply.

Message Severity

ID ATTR_MSGSEV

Type Integer

Description

Message severity. Values range from 00 to 99. The higher the value, the more severe or important the condition.

Multi-item Reply Capability

ID ATTR_MULTI_ITEM_REPLY

Type String

Description

When this attribute value is set to *YES by the client, the performance of list spooled file operations can be greatly improved. The default value is *NO.

Network Identifier

ID ATTR_NETWORK

Type String

Description

The network identifier of the system where the file was created.

Number of Bytes in Spooled File

ID ATTR_NUMBYTES_SPLF

Type Integer

Description

The total number of bytes available in the stream or spooled file. The value indicates the number of bytes BEFORE any transform of the data takes place. In order to accommodate files of sizes greater than 2**31 - 1 bytes, this value is scaled; the user needs to multiply the value by 10K to get the actual number of bytes. This attribute is not valid for spooled files being viewed in page-at-a-time mode.

Number of Bytes to Read/Write

ID ATTR_NUMBYTES

Type Integer

Description

The number of bytes to read for a read operation, or the number of bytes to write for a write operation. The object action determines how to interpret this attribute.

Number of Files

ID ATTR_NUMFILES

Type Integer

Description

The number of spooled files that exist on the output queue.

Number of Writers Started to Queue

ID ATTR_NUMWRITERS

Type Integer

Description

The number of writer jobs started to the output queue.

Object Extended Attribute

ID ATTR_OBJEXTATTR

Type String

Description

An ″extended″ attribute used by some objects like font resources. This value shows up via WRKOBJ and DSPOBJD commands on the server. The title on a server screen may just indicate ″Attribute″. In the case of object types of font resources, for example, common values are CDEPAG, CDEFNT, and FNTCHRSET.

On Job Queue Status

ID ATTR_ONJOBQSTS

Type String

Description

Whether the writer is on a job queue and therefore is not currently running. The possible values are *YES, *NO.

Open time commands

ID ATTR_OPENCMDS

Type String

Description
Specifies whether user wants SCS open time commands to be inserted into datastream before spool file data. Valid values are *YES, *NO.

Operator Controlled

ID ATTR_OPCNTRL

Type String

Description
Whether users with job control authority are allowed to manage or control the spooled files on this queue. Valid values are *YES, *NO.

Order of Files On Queue

ID ATTR_ORDER

Type String

Description
The order of spooled files on this output queue. Valid values are *FIFO, *JOBNBR.

Output Bin

ID ATTR_OUTPUTBIN

Type Integer

Description
The output bin the printer uses for the printed output. Values range from 1 to 65535. The value *DEVD is encoded as 0.

Output Priority

ID ATTR_OUTPTY

Type String

Description
The priority of the spooled file. The priority ranges from 1 (highest) to 9 (lowest). Valid values are 0-9, where 0 represents *JOB.

Output Queue

ID ATTR_OUTPUT_QUEUE

Type String

Description
The Integrated File System path of the output queue. The format of the Integrated File System path is "/QSYS.LIB/library.LIB/queue.OUTQ" where *library* is the library that contains the output queue and *queue* is the name of the output queue.

Output Queue Status

ID ATTR_OUTQSTS

Type String

Description
The status of the output queue. Valid values are RELEASED, HELD.

Overall Status

ID ATTR_OVERALLSTS

Type Integer

Description
The overall status of the ″logical printer″. ″Logical printer″ refers to printer device, output queue and writer job. Valid values are 1 (unavailable), 2 (powered off or not yet available), 3 (stopped), 4 (message waiting), 5 (held), 6 (stop pending), 7 (hold pending), 8 (waiting for printer), 9 (waiting to start), 10 (printing), 11 (waiting for output queue), 12 (connect pending), 13 (powered off), 14 (unusable), 15 (being serviced), 999 (unknown).

Overflow Line Number

ID ATTR_OVERFLOW

Type Integer

Description
The last line to be printed before the data that is being printed overflows to the next page.

Page At A Time

ID ATTR_PAGE_AT_A_TIME

Type String

Description
Specifies whether the spooled file is to be opened in page-at-a-time mode. Valid values are *YES and *NO.

Page Count Estimated

ID ATTR_PAGES_EST

Type String

Description
Specifies whether the page count is estimated rather than actual. Valid values are *YES and *NO.

Page Definition

ID ATTR_PAGE_DEFINITION

Type String

Description
The Integrated File System path name of the page definition or a special value. If an Integrated File System path is specified the format is ″/QSYS.LIB/library.LIB/pagedef.PAGDFN″ where *library* is the library of the page definition and *pagedef* is the name of the page definition. Valid special values include *NONE.

Page Number

ID ATTR_PAGENUMBER

Type Integer

Description
The number of the page to be read from a spooled file opened in page-at-a-time mode.

Pages Per Side

ID ATTR_MULTIUP

Type Integer

Description

The number of logical pages that print on each side of each physical page when the file is printed. Valid values are 1, 2, 4.

Paper Source 1

ID ATTR_PAPER_SOURCE_1

Type String

Description

The size of the paper in paper source one. If this field is not specified or the value is not valid, the special value of *MFRTYPMDL is used. Valid values are *NONE - there is no paper source one or the paper is manually fed into the printer, *MFRTYPMDL - the paper size suggested by the manufacturer type and model is used, *LETTER (8.5 x 11.0 inches), *LEGAL (8.5 x 14.0 inches), *EXECUTIVE (7.25 x 10.5 inches), *LEDGER (17.0 x 11.0 inches), *A3 (297mm x 420mm), *A4 (210mm x 297mm), *A5 (148mm x 210mm), *B4 (257mm x 364mm), *B5 (182mm x 257mm), *CONT80 (8.0 inches wide with continuous form), *CONT132 (13.2 inches wide with continuous form).

Paper Source 2

ID ATTR_PAPER_SOURCE_2

Type String

Description

The size of the paper in paper source two. If this field is not specified or the value is not valid, the special value of *MFRTYPMDL is used. Valid values are *NONE - there is no paper source two or the paper is manually fed into the printer, *MFRTYPMDL - the paper size suggested by the manufacturer type and model is used, *LETTER (8.5 x 11.0 inches), *LEGAL (8.5 x 14.0 inches), *EXECUTIVE (7.25 x 10.5 inches), *LEDGER (17.0 x 11.0 inches), *A3 (297mm x 420mm), *A4 (210mm x 297mm), *A5 (148mm x 210mm), *B4 (257mm x 364mm), *B5 (182mm x 257mm), *CONT80 (8.0 inches wide with continuous form), *CONT132 (13.2 inches wide with continuous form).

Pel Density

ID ATTR_PELDENSITY

Type String

Description

For font resources only, this value is an encoding of the number of pels (″1″ represents a pel size of 240, ″2″ represents a pel size of 320). Additional values may become meaningful as the server defines them.

Point Size

ID ATTR_POINTSIZE

Type Float

Description

The point size in which this spooled file's text is printed. The special value *NONE will be encoded as 0.

Print Fidelity

ID ATTR_FIDELITY

Type String

Description
The kind of error handling that is performed when printing. Valid values are *ABSOLUTE, *CONTENT.

Print on Both Sides

ID ATTR_DUPLEX

Type String

Description
How the information prints. Valid values are *FORMDF, *NO, *YES, *TUMBLE.

Print Quality

ID ATTR_PRTQUALITY

Type String

Description
The print quality that is used when printing this spooled file. Valid values are *STD, *DRAFT, *NLQ, *FASTDRAFT.

Print Sequence

ID ATTR_PRTSEQUENCE

Type String

Description
Print sequence. Valid values are *NEXT.

Print Text

ID ATTR_PRTTEXT

Type String

Description
The text that is printed at the bottom of each page of printed output and on separator pages. Valid special values include *BLANK and *JOB.

Printer

ID ATTR_PRINTER

Type String

Description
The name of the printer device.

Printer Assigned

ID ATTR_PRTASSIGNED

Type String

Description
Indicates if the printer is assigned. Valid values are 1 (assigned to a specific printer), 2 (assigned to multiple printers), 3 (not assigned).

Printer Device Type

ID ATTR_PRTDEVTYPE

Type String

Description
The printer data stream type. Valid values are *SCS, *IPDS, *USERASCII, *AFPDS, *LINE.

Printer File

ID ATTR_PRINTER_FILE

Type String

Description
The Integrated File System path of the printer file. The format of the Integrated File System path is "/QSYS.LIB/library.LIB/printerfile.FILE" where *library* is the library that contains the printer file and *printerfile* is the name of the printer file.

Printer Queue

ID ATTR_RMTPRTQ

Type String

Description
The name of the destination printer queue when sending spooled files via SNDTCPSPLF (LPR).

Publishing Info Color Supported

ID ATTR_PUBINF_COLOR_SUP

Type String

Description
Indicates color is supported for this publishing list entry.

Publishing Info Pages per Minute (Color)

ID ATTR_PUBINF_PPM_COLOR

Type Integer

Description
The pages per minute supported in color mode for this publishing list entry.

Publishing Info Pages per Minute (Monochrome)

ID ATTR_PUBINF_PPM

Type Integer

Description
The pages per minute supported in monochrome for this publishing list entry.

Publishing Info Duplex Support

ID ATTR_PUBINF_DUPLEX_SUP

Type String

Description
The duplex supported indicator for this publishing list entry.

Publishing Info Location

ID ATTR_PUBINF_LOCATION

Type String

Description
The location description for this publishing list entry.

Remote Location Name

ID ATTR_RMTLOCNAME

Type String

Description
The printer device location name.

Record Length

ID ATTR_RECLENGTH

Type Integer

Description
Record length.

Reduce Output

ID ATTR_REDUCE

Type String

Description
The manner in which multiple logical pages print on each side of a physical page. Valid values *TEXT or ????.

Remote System

ID ATTR_RMTSYSTEM

Type String

Description
Remote system name. Valid special values include *INTNETADR.

Replace Unprintable Characters

ID ATTR_RPLUNPRT

Type String

Description
Whether characters that cannot be printed are to be replaced with another character. Valid values are *YES or *NO.

Replacement Character

ID ATTR_RPLCHAR

Type String

Description
The character that replaces any unprintable characters.

Restart Printing

ID ATTR_RESTART

Type Integer

Description
Restart printing. Valid values are -1, -2, -3, or the page number to restart at. The value *STRPAGE is encoded as -1, the value *ENDPAGE is encoded as -2, and the value *NEXT is encoded as -3.

Saddle Stitch Number of Staples

ID ATTR_SADDLESTITCH_NUMSTAPLES

Type Integer

Description
The number of staples that are to be applied along the finishing operation axis.

Saddle Stitch Reference

ID ATTR_SADDLESTITCH_REF

Type String

Description
One or more staples are driven into the media along the finishing operation axis, which is positioned at the center of the media parallel to the reference edge. Valid values are *NONE, *DEVD, *TOP, and *LEFT.

Save Spooled File

ID ATTR_SAVESPLF

Type String

Description
Whether the spooled file is to be saved after it is written. Valid values are *YES, *NO.

Seek Offset

ID ATTR_SEEKOFF

Type Integer

Description
Seek offset. Allows both positive and negative values relative to the seek origin.

Seek Origin

ID ATTR_SEEKORG

Type Integer

Description
Valid values include 1 (beginning or top), 2 (current), and 3 (end or bottom).

Send Priority

ID ATTR_SENDPTY

Type String

Description
Send priority. Valid values are *NORMAL, *HIGH.

Separator page

ID ATTR_SEPPAGE

Type String

Description
Allows a user the option of printing a banner page or not. Valid values are *YES or *NO.

Source Drawer

ID ATTR_SRCDRWR

Type Integer

Description
The drawer to be used when the automatic cut sheet feed option is selected. Valid values are -1, -2, 1-255. The value *E1 is encoded as -1, and the value *FORMDF is encoded as -2.

Spool SCS

ID ATTR_SPLSCS

Type Long

Description
Determines how SCS data is used during create spool file.

Spool the Data

ID ATTR_SPOOL

Type String

Description
Whether the output data for the printer device is spooled. Valid values are *YES, *NO.

Spooled File Creation Authentication Method

ID ATTR_SPLF_AUTH_METHOD

Type Integer

Description
Indicates the client authentication method used to create this spooled file. Valid values include x'00'(*NONE), x'01'(*REQUESTER), x'02'(*BASIC), x'03'(*CERTIFICATE), and 'x'04'(*DIGEST).

Spooled File Creation Security Method

ID ATTR_SPLF_SECURITY_METHOD

Type String

Description
Indicates the security method used to create this spooled file. Valid values are x'00'(*NONE), x'01'(*SSL3), and x'02'(*TLS).

Spooled File Name

ID ATTR_SPOOLFILE

Type String

Description
The name of the spooled file.

Spooled File Number

ID ATTR_SPLFNUM

Type Integer

Description
The spooled file number. Special values allowed are -1 and 0. The value *LAST is encoded as -1, the value *ONLY is encoded as 0.

Spooled File Status

ID ATTR_SPLFSTATUS

Type String

Description
The status of the spooled file. Valid values are *CLOSED, *HELD, *MESSAGE, *OPEN, *PENDING, *PRINTER, *READY, *SAVED, *WRITING.

Spooled Output Schedule

ID ATTR_SCHEDULE

Type String

Description
Specifies, for spooled files only, when the spooled file is available to the writer. Valid values are *IMMED, *FILEEND, *JOBEND.

Started By User

ID ATTR_STARTEDBY

Type String

Description
The name of the user who started the writer.

Starting Page

ID ATTR_STARTPAGE

Type Integer

Description
The page number at which to start printing the spooled file. Valid values are -1, 0, 1, or the page number. The value *ENDPAGE is encoded as -1. For the value 0, printing starts on page 1. For the value 1, the entire file prints.

System Where Created

ID ATTR_SYSTEM

Type String

Description
The name of the system where the spooled file was created. When the name of the system where this spooled file was created cannot be determined, the receiving system name is used.

Text Description

ID ATTR_DESCRIPTION

Type String

Description
Text to describe an instance of an AS400 object.

Time File Opened

ID ATTR_TIMEOPEN

Type String

Description
For spooled files this is the time this spooled file was opened. For AFP resources this is the time the object was last modified. The time is encoded in a character string with the following format, HH MM SS.

Time Spooled File Create Job End

ID ATTR_TIME_END

Type String

Description
The time the job that created the spooled file on the system ended. This field must be set to blanks when special value *ALL is used for field Starting spooled file create date or when special value *LAST is used for field Ending spooled file create date. This field must have a value set if a date is specified for field Ending spooled file create date. The time must be in the HHMMSS format, defined as follows:

- HH - Hour
- MM - Minutes
- SS - Seconds

Time Writer Began Processing Spooled File

ID ATTR_TIME_WTR_BEGAN_FILE

Type String

Description
Indicates the time at which the writer began processing the spooled file. The time is encoded in a character string with the following format, HH MM SS.

Time Writer Completed Processing Spooled File

ID ATTR_TIME_WTR_CMPL_FILE

Type String

Description
Indicates the time at which the writer finished processing the spooled file. The time is encoded in a character string with the following format, HH MM SS.

Total Pages

ID ATTR_PAGES

Type Integer

Description
The number of pages that are contained in a spooled file.

Transform SCS to ASCII

ID ATTR_SCS2ASCII

Type String

Description
Whether the print data is to be transformed from SCS to ASCII. Valid values are *YES, *NO.

Unit of Measure

ID ATTR_UNITOFMEAS

Type String

Description
The unit of measure to use for specifying distances. Valid values are *CM, *INCH.

User Comment

ID ATTR_USERCMT

Type String

Description
The 100 characters of user-specified comment that describe the spooled file.

User Data

ID ATTR_USERDATA

Type String

Description
The 10 characters of user-specified data that describe the spooled file. Valid special values include *SOURCE.

User Defined Data

ID ATTR_USRDFNDTA

Type String

Description
User defined data to be utilized by user applications or user specified programs that process spool files. All characters are acceptable. Max size is 255.

User Defined File

ID ATTR_USRDEFFILE

Type String

Description
Whether the spooled file was created using an API. Valid values are *YES, or *NO.

User Defined Object

ID ATTR_USER_DEFINED_OBJECT

Type String

Description
The Integrated File System path of the user defined object to be utilized by user applications that process spool files. If an Integrated File System path the format of the Integrated File System path is "/QSYS.LIB/library.LIB/object.type" where *library* is the name of the library that contains the object or one of the special values %LIBL% or %CURLIB%. *object* is the name of the object and *type* is the object type. Valid values for *type* include "DTAARA", "DTAQ", "FILE", "PSFCFG", "USRIDX", "USRQ" and "USRSPC". The string "*NONE" is used to indicate no user defined object is to be used.

User Defined Option(s)

ID ATTR_USEDFNOPTS

Type String

Description

User defined options to be utilized by user applications that process spool files. Up to 4 options may be specifies, each value is length char(10). All characters are acceptable.

User Driver Program Data

ID ATTR_USRDRVPGMDTA

Type String

Description

User data to be used with the user driver program. All characters are acceptable. Maximum size is 5000 characters.

User Driver Program

ID ATTR_USER_DRIVER_PROG

Type String

Description

The Integrated File System path of the user defined driver program that processes spooled files. The format of the Integrated File System path is ″/QSYS.LIB/library.LIB/program.PGM″ where *library* is the name of the library that contains the program and *program* is the program name. The *library* may be one of the special values %LIBL% and %CURLIB% or a specific library name. The string ″*NONE″ is used to indicate that no driver program is defined.

User ID

ID ATTR_TOUSERID

Type String

Description

User id to whom the spooled file is sent.

User ID Address

ID ATTR_TOADDRESS

Type String

Description

Address of user to whom the spooled file is sent.

User Transform Program

ID ATTR_USER_TRANSFORM_PROG

Type String

Description

The Integrated File System path of the user defined transform program that transforms spool file data before it is processed by the driver program. The format of the Integrated File System path is ″/QSYS.LIB/library.LIB/program.PGM″ where *library* is the name of the library that contains the program and *program* is the program name. The *library* may be one of the special values %LIBL% and %CURLIB% or a specific library name. The string ″*NONE″ is used to indicate that no transform program is defined.

Viewing Fidelity

ID ATTR_VIEWING_FIDELITY

Type String

Description

The processing to take place when viewing a page of spooled file data (in page-at-a-time mode). Valid values are *ABSOLUTE and *CONTENT(default). To process all non-raster data (commands) prior to the current page, *ABSOLUTE is used. For SCS files, *CONTENT is used to process only open time commands plus the current page. For AFPDS files, *CONTENT is used to process the first page of data plus the current page.

VM/MVS Class

ID ATTR_VMMVSCLASS

Type String

Description

VM/MVS class. Valid values are A-Z and 0-9.

Waiting for Data Status

ID ATTR_WTNGDATASTS

Type String

Description

Whether the writer has written all the data currently in the spooled file and is waiting for more data. Possible values are *NO - the writer is not waiting for more data, *YES - the writer has written all the data currently in the spooled file and is waiting for more data. This condition occurs when the writer is producing an open spooled file with SCHEDULE(*IMMED) specified.

Waiting for Device Status

ID ATTR_WTNGDEVSTS

Type String

Description

Whether the writer is waiting to get the device from a job that is printing directly to the printer. Values are *NO - the writer is not waiting for the device, *YES - the writer is waiting for the device.

Waiting for Message Status

ID ATTR_WTNGMSGSTS

Type String

Description

Whether the writer is waiting for a reply to an inquiry message. Values are *NO and *YES.

When to Automatically End Writer

ID ATTR_WTRAUTOEND

Type String

Description

When to end the writer if it is to be ended automatically. Valid values are *NORDYF, *FILEEND. Attribute Automatically end writer must be set to *YES.

When to End Writer

ID ATTR_WTREND

Type String

Description
When to end the writer. Valid value are *CNTRLD, *IMMED, and *PAGEEND. This is different from when to automatically end the writer.

When to Hold File

ID ATTR_HOLDTYPE

Type String

Description
When to hold the spooled file. Valid values are *IMMED, and *PAGEEND.

Width of Page

ID ATTR_PAGEWIDTH

Type Float

Description
The width of a page. Units of measurement are specified in the measurement method attribute.

Workstation Customizing Object

ID ATTR_WORKSTATION_CUST_OBJECT

Type String

Description
The Integrated File System path of the workstation customizing object. The format of the Integrated File System path is "/QSYS.LIB/library.LIB/custobj.WSCST" where *library* is the library that contains the customization object and *custobj* is the name of the workstation customization object.

Writer Job Name

ID ATTR_WRITER

Type String

Description
The name of the writer job.

Writer Job Number

ID ATTR_WTRJOBNUM

Type String

Description
The writer job number.

Writer Job Status

ID ATTR_WTRJOBSTS

Type String

Description
The status of the writer job. Valid values are STR, END, JOBQ, HLD, MSGW.

Writer Job User Name

ID ATTR_WTRJOBUSER

Type String

Description
The name of the user that started the writer job.

Writer Started

ID ATTR_WTRSTRTD

Type String

Description
Indicates whether a writer is started for this printer. Values are 1 - yes a writer is started, 0 - no writer is started.

Writer Starting Page

ID ATTR_WTRSTRPAGE

Type Integer

Description
Specifies the page number of the first page to print from the first spooled file when the writer job starts. This is only valid if the spooled file name is also specified when the writer starts.

Writing Status

ID ATTR_WRTNGSTS

Type String

Description
Indicates whether the print writer is in writing status. Values are *YES - the writer is in writing status, *NO - the writer is not in writing status, *FILE - the writer is writing the file separators.

Network Print Server Object Attributes

NPS CCSID

ID ATTR_NPSCCSID

Type Integer

Description
CCSID that the Network Print Server expects that all strings will be encoded in.

NPS Level

ID ATTR_NPSLEVEL

Description
The version, release, and modification level of the Network Print Server. This attribute is a character string encoded as VXRYMY (ie. ″V3R1M0″) where

```
X is in (0..9)
Y is in (0..9,A..Z)
```

Copying spooled files:

You can use the copy method of the SpooledFile class to create a copy of the spooled file that the SpooledFile object represents. Using SpooledFile.copy() performs the following actions:

- Creates the new spooled file on the same output queue and on the same system as the original spooled file
- Returns a reference to the new spooled file

SpooledFile.copy() is a new method available to you only if you download JTOpen 3.2 or later or apply an i5/OS fix. It is recommended that the better solution is to download and use JTOpen. For more information, see the following:

IBM Toolbox for Java and JTOpen: Downloads 

IBM Toolbox for Java and JTOpen: Service Packs 

The copy method uses the Create Spooled File (QSPCRTSP) API within the network print server job to create an exact replica of the spooled file. You need only a unique creation date and time to preserve the identity of the newly created copy of the spooled file. For more information about the QSPCRTSP API, see the following information:

Create Spooled File (QSPCRTSP) API

Specifying an output queue as a parameter to the copy method creates the copy of the spooled file to the first position on the specified output queue. Both the output queue and the original spooled file must reside on the same system

Example: Copying a spooled file using SpooledFile.copy()

Note: Read the Code example disclaimer for important legal information.

This example shows how to use SpooledFile.copy() to copy a spooled file to the same queue that contains the file you want to copy. When you want to route the newly copied spooled file to a specific output queue, pass the output queue as a parameter to the copy method:

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

where <outqname> is the OutputQueue object.

```
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 iSeries 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){
}
```

Javadoc reference documentation

For more information about SpooledFile.copy(), see the following Javadoc reference documentation:

The SpooledFile copy() method

Creating new spooled files:

You can use the SpooledFileOutputStream class to create new server spooled files. The class derives from the standard JDK java.io.OutputStream class; after its construction, it can be used anywhere an OutputStream is used.

When creating a new SpooledFileOutputStream, the caller may specify the following:

- Which printer file to use
- Which output queue to put the spooled file on
- A PrintParameterList object that may contain parameters to override fields in the printer file

These parameters are all optional (the caller may pass null of any or all of them). If a printer file is not specified, the network print server uses the default network print printer file, QPNPSPRTF. The output queue parameter is there as a convenience; it also can be specified in the PrintParameterList. If the output queue parameter is specified in both places, the PrintParameterList field overrides the output queue parameter. See the documentation of the SpooledFileOutputStream constructor for a complete list of which attributes may be set in the PrintParameterList for creating new spooled files.

Use one of the write() methods to write data into the spooled file. The SpooledFileOutputStream object buffers the data and sends it when either the output stream is closed or the buffer is full. Buffering is done for two reasons:

- It allows the automatic data typing (see Data stream types in spooled files) to analyze a full-buffer of data to determine the data type
- It makes the output stream work faster because not every write request is communicated to the server.

Use the flush() method to force the data to be written to the server.

When the caller is finished writing data to the new spooled file, the close() method is called to close the spooled file. Once the spooled file has been closed, no more data can be written to it. By calling the getSpooledFile() method once the spooled file has been closed, the caller can get a reference to a SpooledFile object that represents the spooled file.

Data stream types in spooled files

Use the Printer Data Type attribute of the spooled file to set the type of data to be put into the spooled file. If the caller does not specify a printer data type, the default is to use automatic data typing. This method looks at the first few thousand bytes of the spooled file data, determines if it fits either SNA Character Stream (SCS) or Advanced Function Printing data stream (AFPDS) data stream architectures, and then sets the attribute appropriately. If the bytes of spooled file data do not match either of these architectures, the data is tagged as *USERASCII. Automatic data typing works most of the time. The caller generally uses it unless the caller has a specific case in which automatic data typing does not work. In those cases, the caller can set the Printer Data Type attribute to a specific value (for example, *SCS). If the caller wants to use the printer data that is in the printer file, the caller must use the special value *PRTF. If the caller overrides the default data type when creating a spooled file, caution must be used to ensure that the data put into the spooled file matches the data type attribute. Putting non-SCS data into a spooled file that is marked to receive SCS data triggers an error message from the host and the loss of the spooled file.

Generally, this attribute can have three values:

- ***SCS** - an EBCDIC, text-based printer data stream.
- ***AFPDS** (Advanced Function Presentation™ Data Stream) - another data stream supported on the server. *AFPDS can contain text, image, and graphics, and can use external resources such as page overlays and external images in page segments.
- ***USERASCII** - any non-SCS and non-AFPDS printer data that the server handles by just passing it through. Postscript and HP-PCL data streams are examples data streams that are in a *USERASCII spooled file.

Examples

The following examples show ways you can work with spooled files. The first example shows how to create a spooled file on a server from an input stream. The second example shows how to generate an SCS data stream using the SCS3812Writer class, and how to write the stream to a spooled file on the server.

"Example: Creating spooled files" on page 487

"Example: Creating SCS spooled files" on page 488

Generating an SCS data stream:

To generate spooled files that will print on certain printers attached to the server, an SNA Character Stream (SCS) data stream must be created. (SCS is a text-based, EBCDIC data stream that can be printed on SCS printers, IPDS™ printers, or to PC printers.) SCS can be printed by converting it using an emulator or the host print transform on the server.

You can use the SCS writer classes to generate such an SCS data stream. The SCS writer classes convert Java Unicode characters and formatting options into an SCS data stream. Five SCS writer classes generate varying levels of SCS data streams. The caller chooses the writer that matches the final printer destination to which the caller or end user will be printing.

Use the following SCS writer classes to generate an SCS printer data stream:

SCS writer class	Description
SCS5256Writer	The simplest SCS writer class. Supports text, carriage return, line feed, new line, form feed, absolute horizontal and vertical positioning, relative horizontal and vertical positioning, and set vertical format.
SCS5224Writer	Extends the 5256 writer and adds methods to set character per inch (CPI) and lines per inch (LPI).
SCS5219Writer	Extends the 5224 writer and adds support for left margin, underline, form type (paper or envelope), form size, print quality, code page, character set, source drawer number, and destination drawer number.
SCS5553Writer	Extends the 5219 writer and adds support for character rotation, grid lines, and font scaling. The 5553 is a double-byte character set (DBCS) data stream.
SCS3812Writer	Extends the 5219 writer and adds support for bold, duplex, text orientation, and fonts.

To construct an SCS writer, the caller needs an output stream and, optionally, an encoding. The data stream is written to the output stream. To create an SCS spooled file, the caller first constructs a SpooledFileOutputStream, and then uses that to construct an SCS writer object. The encoding parameter gives a target EBCDIC coded character set identifier (CCSID) to convert the characters to.

Once the writer is constructed, use the write() methods to output text. Use the carriageReturn(), lineFeed(), and newLine() methods to position the write cursor on the page. Use the endPage() method to end the current page and start a new page.

When all of the data has been written, use the close() method to end the data stream and close the output stream.

Example

The following example shows how to generate a SCS data stream using the SCS3812Writer class, and how to write the stream to a spooled file on the server:

Example: Creating SCS spooled files

Reading spooled files and AFP resources:

You can use the PrintObjectInputStream class to read the raw contents of a spooled file or Advanced Function Printing (AFP) resource from the server. The class extends the standard JDK java.io.InputStream class so that it can be used anywhere an InputStream is used.

Obtain a PrintObjectInputStream object by calling either the getInputStream() method on an instance of the SpooledFile class or the getInputStream() method on an instance of the AFPResource class. Getting an input stream for a spooled file is supported for Version 3 Release 2 (V3R2), V3R7, and later versions of i5/OS. Getting input streams for AFP resources is supported for V3R7 and later.

Use one of the read() methods for reading from the input stream. These methods all return the number of bytes actually read, or -1 if no bytes were read and the end of file was reached.

Use the available() method of PrintObjectInputStream to return the total number of bytes in the spooled file or AFP resource. The PrintObjectInputStream class supports marking the input stream, so PrintObjectInputStream always returns true from the markSupported() method. The caller can use the mark() and reset() methods to move the current read position backward in the input stream. Use the skip() method to move the read position forward in the input stream without reading the data.

Example

The following example shows how to use PrintObjectInputStream to read an existing server spooled file

Example: Reading spooled files

Reading spooled files using PrintObjectPageInputStream and PrintObjectTransformedInputStream:

You can use the PrintObjectPageInputStream class to read the data out of a server AFP and SCS spooled file one page at a time.

You can obtain a PrintObjectPageInputStream object with the getPageInputStream() method.

Use one of the read() methods for reading from the input stream. All these methods return the number of bytes actually read, or -1 if no bytes were read and the end of page was reached.

Use the available() method of PrintObjectPageInputStream to return the total number of bytes in the current page. The PrintObjectPageInputStream class supports marking the input stream, so PrintObjectPageInputStream always returns true from the markSupported() method. The caller can use the mark() and reset() methods to move the current read position backward in the input stream so that subsequent reads reread the same bytes. The caller can use the skip() method to move the read position forward in the input stream without reading the data.

However, when transforming an entire spooled file data stream is desired, use the PrintObjectTransformedInputStream class.

Example

The following example shows how to use PrintObjectPageInputStream and PrintObjectTransformedInputStream to obtain different transformations when reading spooled file data:

"Example: Reading and transforming spooled files" on page 491

Product license

The ProductLicense class enables you to request licenses for products installed on the iSeries. To be compatible with other iSeries license users, the class works through iSeries product license support when requesting or releasing a license.

The class does not enforce the license policy but returns enough information such that the application can enforce the policy. When a license is requested the ProductLicense class will return the status of the

request -- license granted or denied. If the request is denied the application must disable the behavior that required the license because the IBM Toolbox for Java does not know which function to disable.

Use the ProductLicense class with iSeries license support to enforce the license of your application:

- The server side of your application registers your product and license terms with iSeries license support.
- The client side of your application uses the ProductLicense object to request and release licenses.

Example: ProductLicense scenario

For example, suppose your customer bought 15 concurrent use licenses for your product. Concurrent use means 15 users can use the product at the same time, but it does not need to be 15 specific users. It can be any 15 users in the organization. This information is registered with iSeries license support. As users connect your application uses the ProductLicense class to request a license.

- When the number of concurrent users is fewer than 15, the request is successful and your application runs.
- When the 16th user connects, the ProductLicense request fails. Your application then displays an error message and terminates.

When a user stops running the application, your application releases the license by way of the ProductLicense class. The license is now available for someone else to use.

For more information and a code example, refer to the ProductLicense javadoc.

ProgramCall class

The ProgramCall class allows the Java program to call an iSeries program. You can use the ProgramParameter class to specify input, output, and input/output parameters. If the program runs, the output and input/output parameters contain the data that is returned by the iSeries program. If the iSeries program fails to run successfully, the Java program can retrieve any resulting iSeries messages as a list of AS400Message objects.

Required parameters are as follows:

- The program and parameters to run
- The AS400 object that represents the iSeries server that has the program.

The program name and parameter list can be set on the constructor, through the setProgram() method, or on the run() method The run() method calls the program.

Using the ProgramCall class causes the AS400 object to connect to the iSeries server. See managing connections for information about managing connections.

Example: Using ProgramCall

Note: Read the Code example disclaimer for important legal information.

The following example shows how to use the ProgramCall class:

```
                    // 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);
```

The ProgramCall object requires the integrated file system path name of the program.

The default behavior is for iSeries programs to run in a separate server job, even when the Java program and the iSeries program are on the same server. You can override the default behavior and have the iSeries program run in the Java job using the setThreadSafe() method.

Using ProgramParameter objects

You can use the ProgramParameter objects to pass parameter data between the Java program and the iSeries program. Set the input data with the setInputData() method. After the program is run, retrieve the output data with the getOutputData() method. Each parameter is a byte array. The Java program must convert the byte array between Java and iSeries formats. The data conversion classes provide methods for converting data. Parameters are added to the ProgramCall object as a list.

Example: Using ProgramParameter

Note: Read the Code example disclaimer for important legal information.

The following example shows how to use the ProgramParameter object to pass parameter data.

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

                    // My program has two parameters.
                    // Create a list to hold these
                    // parameters.
ProgramParameter[] parmList = new ProgramParameter[2];

                    // First parameter is an input
                    // parameter
byte[] key = {1, 2, 3};
parmList[0] = new ProgramParameter(key);

                    // Second parameter is an output
                    // parameter. A four-byte number
                    // is returned.
parmList[1] = new ProgramParameter(4);

                    // Create a program object
                    // specifying the name of the
                    // program and the parameter list.
```

```
ProgramCall pgm = new ProgramCall(sys, "/QSYS.LIB/MYLIB.LIB/MYPROG.PGM", parmList);

                  // Run the program.
if (pgm.run() != true)
{

                  // If the iSeries cannot run the
                  // program, look at the message list
                  // to find out why it didn't run.
   AS400Message[] messageList = pgm.getMessageList();

}
else
{
                  // Else the program ran. Process the
                  // second parameter, which contains
                  // the returned data.

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

                  // Convert from iSeries type to Java
                  // object. The number starts at the
                  // beginning of the buffer.
   byte[] data = parmList[1].getOutputData();
   int i = bin4Converter.toInt(data);
}

                  // Disconnect since I am done
                  // running programs
sys.disconnectService(AS400.COMMAND);
```

QSYSObjectPathName class

You can use the QSYSObjectPathName class to represent an object in the integrated file system. Use this class to build an integrated file system name or to parse an integrated file system name into its components.

Several of the IBM Toolbox for Java classes require an integrated file system path name in order to be used. Use a QSYSObjectPathName object to build the name.

Note: Read the Code example disclaimer for important legal information.

The following examples show how to use the QSYSObjectPathName class:

Example 1: The ProgramCall object requires the integrated file system name of the server program to call. A QSYSObjectPathName object is used to build the name. To call program PRINT_IT in library REPORTS using a QSYSObjectPathName:

```
                  // 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
```

Example 2: If the name of the AS400 object is used just once, the Java program can use the toPath() method to build the name. This method is more efficient than creating a QSYSObjectPathName object.

```
                    // 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
```

Example 3: In this example, a Java program was given an integrated file system path. The QSYSObjectPathName class can be used to parse this name into its components:

```
                    // 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();
```

Record-level access

The record-level access classes provide the ability to do the following:

- Create an iSeries physical file specifying one of the following:
  - The record length
  - An existing data description specifications (DDS) source file
  - A RecordFormat object
- Retrieve the record format from an iSeries physical or logical file, or the record formats from an iSeries multiple format logical file.

  Note: The record format of the file is not retrieved in its entirety. The record formats retrieved are meant to be used when setting the record format for an AS400File object. Only enough information is retrieved to describe the contents of a record of the file. Record format information, such as column headings and aliases, is not retrieved.
- Access the records in an iSeries file sequentially, by record number, or by key.
- Write records to an iSeries file.
- Update records in an iSeries file sequentially, by record number, or by key.
- Delete records in an iSeries file sequentially, by record number, or by key.
- Lock an iSeries file for different types of access.
- Use commitment control to allow a Java program to do the following:

  - Start commitment control for the connection.
  - Specify different commitment control lock levels for different files.
  - Commit and rollback transactions.
- Delete iSeries files.
- Delete a member from an iSeries file.

Note: The record-level access classes do not support logical join files or null key fields.

The following classes perform these functions:
- The AS400File class is the abstract base class for the record-level access classes. It provides the methods for sequential record access, creation and deletion of files and members, and commitment control activities.
- The KeyedFile class represents an iSeries file whose access is by key.
- The SequentialFile class represents an iSeries file whose access is by record number.
- The AS400FileRecordDescription class provides the methods for retrieving the record format of an iSeries file.

The record-level access classes require an AS400 object that represents the system that has the database files. Using the record-level access classes causes the AS400 object to connect to the iSeries. See managing connections for information about managing connections.

The record-level access classes require the integrated file system path name of the data base file. See integrated file system path names for more information.

The record-level access classes use the following:
- The RecordFormat class to describe a record of the database file
- The Record class to provide access to the records of the database file
- The LineDataRecordWriter class to write a record in line data format

These classes are described in the data conversion section.

Examples
- The sequential access example shows how to access an iSeries file sequentially.
- The read file example shows how to use the record-level access classes to read an iSeries file.
- The keyed file example shows to to use the record-level access classes to read records by key from an iSeries file.

AS400File:

The AS400File class provides the methods for the following:
- Creating and deleting server physical files and members
- Reading and writing records in server files
- Locking files for different types of access
- Using record blocking to improve performance
- Setting the cursor position within an open server file
- Managing commitment control activities

KeyedFile:

The KeyedFile class gives a Java program keyed access to a file on the server. Keyed access means that the Java program can access the records of a file by specifying a key. Methods exist to position the cursor, read, update, and delete records by key.

To position the cursor, use the following methods:

- positionCursor(Object[]) - set cursor to the first record with the specified key.
- positionCursorAfter(Object[]) - set cursor to the record after the first record with the specified key.
- positionCursorBefore(Object[]) - set cursor to the record before the first record with the specified key.

To delete a record, use the following method :

- deleteRecord(Object[]) - delete the first record with the specified key.

The read methods are:

- read(Object[]) - read the first record with the specified key.
- readAfter(Object[]) - read the record after the first record with the specified key.
- readBefore(Object[]) - read the record before the first record with the specified key.
- readNextEqual() - read the next record whose key matches the specified key. Searching starts from the record after the current cursor position.
- readPreviousEqual() - read the previous record whose key matches the specified key. Searching starts from the record before the current cursor position.

To update a record, use the following method:

- update(Object[]) - update the record with the specified key.

Methods are also provided for specifying a search criteria when positioning, reading, and updating by key. Valid search criteria values are as follows:

- Equal - find the first record whose key matches the specified key.
- Less than - find the last record whose key comes before the specified key in the key order of the file.
- Less than or equal - find the first record whose key matches the specified key. If no record matches the specified key, find the last record whose key comes before the specified key in the key order of the file.
- Greater than - find the first record whose key comes after the specified key in the key order of the file.
- Greater than or equal - find the first record whose key matches the specified key. If no record matches the specified key, find the first record whose key comes after the specified key in the key order of the file.

KeyedFile is a subclass of AS400File; all methods in AS400File are available to KeyedFile.

Specifying the key

The key for a KeyedFile object is represented by an array of Java Objects whose types and order correspond to the types and order of the key fields as specified by the RecordFormat object for the file.

The following example shows how to specify the key for the KeyedFile object.

```
            // 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";
```

A KeyedFile object accepts partial keys as well as complete keys. However, the key field values that are specified must be in order.

For example:

```
            // 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";
```

Null keys and null key fields are not supported.

The key field values for a record can be obtained from the Record object for a file through the getKeyFields() method.

The following example shows how to read from a file by key:

```
            // 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);
```

SequentialFile:

The SequentialFile class gives a Java program access to a file on the server by record number. Methods exist to position the cursor, read, update, and delete records by record number.

To position the cursor, use the following methods:

- positionCursor(int) - set cursor to the record with the specified record number.
- positionCursorAfter(int) - set cursor to the record after the specified record number.
- positionCursorBefore(int) - set cursor to the record before the specified record number.

To delete a record, use the following method:

- deleteRecord(int) - delete the record with the specified record number.

To read a record, use the following methods:

- read(int) - read the record with the specified record number.
- readAfter(int) - read the record after the specified record number.
- readBefore(int) - read the record before the specified record number.

To update a record, use the following method:

- update(int) - update the record with the specified record number.

SequentialFile is a subclass of AS400File; all methods in AS400File are available to SequentialFile.

The following example shows how to use the SequentialFile class:

```
                    // 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);
```

AS400FileRecordDescription:

The AS400FileRecordDescription class provides the methods for retrieving the record format of a file on the server. This class provides methods for creating Java source code for subclasses of RecordFormat and for returning RecordFormat objects, which describe the record formats of user-specified physical or logical files on the server. The output of these methods can be used as input to an AS400File object when setting the record format.

It is recommended that the AS400FileRecordDescription class always be used to generate the RecordFormat object when the file already exists on the server.

Note: The AS400FileRecordDescription class does not retrieve the entire record format of a file. Only enough information is retrieved to describe the contents of the records that make up the file. Information such as column headings, aliases, and reference fields is not retrieved. Therefore, the record formats retrieved cannot necessarily be used to create a file whose record format is identical to the file from which the format was retrieved.

Creating Java source code for subclasses of RecordFormat to represent the record format of files on the server

The createRecordFormatSource() method creates Java source files for subclasses of the RecordFormat class. The files can be compiled and used by an application or applet as input to the AS400File.setRecordFormat() method.

The createRecordFormatSource() method should be used as a development time tool to retrieve the record formats of existing files on the server. This method allows the source for the subclass of the RecordFormat class to be created once, modified if necessary, compiled, and then used by many Java programs accessing the same files on the server. Because this method creates files on the local system, it can be used only by Java applications. The output (the Java source code), however, can be compiled and then used by Java applications and applets alike.

Note: This method overwrites files with the same names as the Java source files being created.

Example 1: The following example shows how to use the createRecordFormatSource() method:

```
                  // 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.
```

Example 2: Compile the file you created above, FILE1Format.java, and use it as follows:

```
                    // 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);
```

Creating RecordFormat objects to represent the record format of files on the server

The retrieveRecordFormat() method returns an array of RecordFormat objects that represent the record formats of an existing file on the server. Typically, only one RecordFormat object is returned in the array. When the file for which the record format is being retrieved is a multiple format logical file, more than one RecordFormat object is returned. Use this method to dynamically retrieve the record format of an existing file on the server during runtime. The RecordFormat object then can be used as input to the AS400File.setRecordFormat() method.

The following example shows how to use the retrieveRecordFormat() method:

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

                    // Create an AS400FileRecordDescription object that represents the file
AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys,
                                    "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");
                    // Retrieve the record format for the file
RecordFormat[] format = myFile.retrieveRecordFormat();

                    // Create an AS400File object that represents the file
SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Set the record format
myFile.setRecordFormat(format[0]);

                    // Open the file and read from it
                    ....

                    // Close the file since I am done using it
```

```
myFile.close();

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

Creating and deleting files and members:

Physical files on the server are created by specifying a record length, an existing server data description specifications (DDS) source file, or a RecordFormat object.

When you create a file and specify a record length, a data file or a source file can be created. The method sets the record format for the object. Do not call the setRecordFormat() method for the object.

A data file has one field. The field name is the name of the file, the field type is of type character, and the field length is the length that is specified on the create method.

A source file has three fields:

- Field SRCSEQ is ZONED DECIMAL (6,2)
- Field SRCDAT is ZONED DECIMAL (6,0)
- SRCDTA is a character field with a length that is the length specified on the create method minus 12

The following examples show how to create files and members.

Example 1: To create a data file with a 128-byte record:

```
                  // 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);
```

Example 2: When creating a file specifying an existing DDS source file, the DDS source file is specified on the create() method. The record format for the file must be set using the setRecordFormat() method before the file can be opened. For example:

```
                  // 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);
```

Example 3: When creating a file specifying a RecordFormat object, the RecordFormat object is specified on the create() method. The method sets the record format for the object. The setRecordFormat() method must not be called for the object.

```
                    // 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);
```

When deleting files and members, use these methods:

- Use the delete() method to delete server files and all of their members.
- Use the deleteMember() method to delete just one member of a file.

Use the addPhysicalFileMember() method to add members to a file.

Reading and writing records:

You can use the AS400File class to read, write, update, and delete records in files on the server. The record is accessed through the Record class, which is described by a RecordFormat class. The record format must be set through the setRecordFormat() method before the file is opened, unless the file was just created (without an intervening close()) by one of the create() methods, which sets the record format for the object.

Use the read() methods to read a record from the file. Methods are provided to do the following:

- read() - read the record at the current cursor position
- readFirst() - read the first record of the file
- readLast() - read the last record of the file
- readNext() - read the next record in the file
- readPrevious() - read the previous record in the file

The following example shows how to use the readNext() method:

```
                  // 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);
```

Use the update() method to update the record at the cursor position.

For example:

```
                  // 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);
```

Use the write() method to append records to the end of a file. A single record or an array of records can be appended to the file.

Use the deleteCurrentRecord() method to delete the record at the cursor position.

Locking files:

The Java program can lock a file to prevent other users from accessing the file while the first Java program is using the file. Lock types are as follows:

- Read/Exclusive Lock - The current Java program reads records, and no other program can access the file.
- Read/Allow shared read Lock - The current Java program reads records, and other programs can read records from the file.

- Read/Allow shared write Lock - The current Java program reads records, and other programs can change the file.
- Write/Exclusive Lock - The current Java program changes the file, and no other program can access the file.
- Write/Allow shared read Lock - The current Java program changes the file, and other programs can read records from the file.
- Write/Allow shared write Lock - The current Java program changes the file, and other programs can change the file.

To give up the locks obtained through the lock() method, the Java program starts the releaseExplicitLocks() method.

Using record blocking:

The AS400File class uses record blocking to improve performance:

- If the file is opened for read-only access, a block of records is read when the Java program reads a record. Blocking improves performance because subsequent read requests may be be handled without accessing the server. Little performance difference exists between reading a single record and reading several records. Performance improves significantly if records can be served out of the block of records cached on the client.

  The number of records to read in each block can be set when the file is opened. For example:

```
                    // 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);
```

- If the file is opened for write-only access, the blocking factor indicates how many records are written to the file at one time when the write(Record[]) method is invoked.

  For example:

```
                    // 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);
```

- If the file is opened for read-write access, no blocking is done. Any blocking factor specified on open() is ignored.

Setting the cursor position:

An open file has a cursor. The cursor points to the record to be read, updated, or deleted. When a file is first opened the cursor points to the beginning of the file. The beginning of the file is before the first record. Use the following methods to set the cursor position:

- positionCursorAfterLast() - Set cursor to after the last record. This method exists so Java programs can use the readPrevious() method to access records in the file.
- positionCursorBeforeFirst() - Set cursor to before the first record. This method exists so Java programs can use the readNext() method to access records in the file.
- positionCursorToFirst() - Set the cursor to the first record.
- positionCursorToLast() - Set the cursor to the last record.
- positionCursorToNext() - Move the cursor to the next record.
- positionCursorToPrevious() - Move the cursor to the previous record.

The following example shows how to use the positionCursorToFirst() method to position the cursor.

```
                    // 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);
```

Commitment control:

Through commitment control, your Java program has another level of control over changing a file. With commitment control turned on, transactions to a file are pending until they are either committed or rolled back. If committed, all changes are put to the file. If rolled back, all changes are discarded. The transaction can be changing an existing record, adding a record, deleting a record, or even reading a record depending on the commitment control lock level specified on the open().

The levels of commitment control are as follows:

- All - Every record accessed in the file is locked until the transaction is committed or rolled back.
- Change - Updated, added, and deleted records in the file are locked until the transaction is committed or rolled back.

- Cursor Stability - Updated, added, and deleted records in the file are locked until the transaction is committed or rolled back. Records that are accessed but not changed are locked only until another record is accessed.
- None - There is no commitment control on the file. Changes are immediately put to the file and cannot be rolled back.

You can use the startCommitmentControl() method to start commitment control. Commitment control applies to the AS400 **connection**. Once commitment control is started for a connection, it applies to all files opened under that connection from the time that commitment control was started. Files opened before commitment control is started are not under commitment control. The level of commitment control for individual files is specified on the open() method. Specify COMMIT_LOCK_LEVEL_DEFAULT to use the same level of commitment control as was specified on the startCommitmentControl() method.

For example:

```
                    // 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);
```

The commit() method commits all transactions since the last commit boundary for the **connection**. The rollback() method discards all transactions since the last commit boundary for the **connection**. Commitment control for a connection is ended through the endCommitmentControl() method. If a file is closed before invoking the commit() or rollback() method, all uncommitted transactions are rolled back. All files opened under commitment control must be closed before the endCommitmentControl() method is called.

The following examples shows how to start commitment control, commit or roll back functions, and then end commitment control:

```
                    // ... 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();
```

Service program call

The ServiceProgramCall class allows you to call an iSeries service program. ServiceProgramCall is a subclass of the ProgramCall class that you use to call iSeries programs. If you want to call an iSeries program, use the ProgramCall class.

The ServiceProgramCall class makes it possible for you to call an iSeries service program, pass data to an iSeries service program through input parameters, and access data the iSeries service program returns through output parameters. Using ServiceProgramCall causes the AS400 object to connect to the iSeries. See managing connections for information about managing connections.

The default behavior is for service programs to run in a separate server job, even when the Java program and the service program are on the same server. You can override the default behavior and have the service program run in the Java job using the inherited (from ProgramCall) setThreadSafe() method.

Using the ServiceProgramCall class

In order to use the ServiceProgramCall class, you must be sure to meet the following requirements:

- The service program must be on an iSeries
- You can pass no more than seven parameters to the service program
- The return value of the service program is void or numeric

Working with ProgramParameter objects

The ProgramParameter class works with the ServiceProgramCall class to pass parameter data to and from an iSeries service program. You pass input data to the iSeries service program with setInputData().

You request the amount of output data you want returned with setOutputDataLength(). You retrieve the output data after the service program is finished running with getOutputData(). In addition to the data itself, ServiceProgramCall needs to know how to pass parameter data to the service program. The setParameterType() method of ProgramParameter is used to supply this information. The type indicates if the parameter is pass by value or pass by reference. In either case, data is sent from the client to the server. Once the data is on the iSeries, the server uses the parameter type to correctly call the service program.

All parameters will be in the form of a byte array. Therefore, to convert between iSeries and Java formats, you use the data conversion and description classes.

SystemStatus classes

The SystemStatus classes allow you to retrieve system status information and to retrieve and change system pool information. The SystemStatus object allows you to retrieve system status information including the following:

- getUsersCurrentSignedOn(): Returns the number of users currently signed on the system
- getUsersTemporarilySignedOff(): Returns the number of interactive jobs that are disconnected
- getDateAndTimeStatusGathered(): Returns the date and time when the system status information was gathered
- getJobsInSystem(): Returns the total number of user and system jobs that are currently running
- getBatchJobsRunning(): Returns the number of batch jobs currently running on the system
- getBatchJobsEnding(): Returns the number of batch jobs that are in the process of ending
- getSystemPools(): Returns an enumeration containing a SystemPool object for each system pool

In addition to the methods within the SystemStatus class, you also can access SystemPool through SystemStatus. SystemPool allows you to get information about system pools and change system pool information.

Example

Note: Read the Code example disclaimer for important legal information.

This example shows you how to use caching with the SystemStatus class:

```
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();
```

SystemPool class:

The SystemPool class allows you to retrieve and change system pool information including the following:

- The getPoolSize() method returns the size of the pool, and the setPoolSize() method sets the size of the pool.
- The getPoolName() method retrieves the name of the pool, and the setPoolName() method sets the pool's name.
- The getReservedSize() method returns the amount of storage in the pool that is reserved for system use.
- The getDescription() method returns the description of the system pool.
- The getMaximumActiveThreads() method returns the maximum number of threads that can be active in the pool at any one time.
- The setMaximumFaults() method sets the maximum faults-per-second guideline to use for this system pool.
- The setPriority() method sets the priority of this system pool relative to the priority of the other system pools.

Example

Note: Read the Code example disclaimer for important legal information.

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

System values

The system value classes allow a Java program to retrieve and change system values and network attributes. You can also define your own group to contain the system values you want.

A SystemValue object primarily contains the following information:

- Name
- Description
- Release
- Value

Using the SystemValue class, retrieve a single system value by using the getValue() method and change a system value by using the setValue() method.

You can also retrieve group information about a particular system value:

- To retrieve the system-defined group to which a system value belongs, use the getGroup() method.
- To retrieve the user-defined group to which a SystemValue object belongs (if any), use the getGroupName() and getGroupDescription() methods.

Whenever the value of a system value is retrieved for the first time, the value is retrieved from the iSeries and cached. On subsequent retrievals, the cached value is returned. If the current iSeries value is what you want instead of the cached value, a clear() must be done to clear the current cache.

System value list

SystemValueList represents a list of system values on the specified iSeries server. The list is divided into several system-defined groups that allow the Java program to access a portion of the system values at a time.

System value group

SystemValueGroup represents a user-defined collection of system values and network attributes. Rather than a container, it is instead a factory for generating and maintaining unique collections of system values.

You can create a SystemValueGroup by specifying one of the system-defined groups (one of the constants in the SystemValueList class) or by specifying an array of system value names.

You can individually add the names of system values to include in the group by using the add() method. You can also remove them by using the remove() method.

Once the SystemValueGroup is populated with the required system value names, obtain the real SystemValue objects from the group by calling the getSystemValues() method. In this way, a SystemValueGroup object takes a set of system value names and generates a Vector of SystemValue objects, all having the system, group name, and group description of the SystemValueGroup.

To refresh a Vector of SystemValue objects all at once, use the refresh() method.

Examples of using the SystemValue and SystemValueList classes

Note: Read the Code example disclaimer for important legal information.

The following example shows how to create and retrieve a system value:

```
//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();
```

Examples of using the SystemValueGroup class

The following example shows how to build a group of system value names and then manipulate them:

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

//Create a system value group initially representing all of the network attributes on the system.
String name = "My Group";
String description = "This is one of my system values.";
SystemValueGroup svGroup = new SystemValueGroup(sys, name, description, SystemValueList.GROUP_NET);

//Add some more system value names to the group and remove some we do not want.
svGroup.add("QDATE");
svGroup.add("QTIME");
svGroup.remove("NETSERVER");
svGroup.remove("SYSNAME");

//Obtain the actual SystemValue objects. They are returned inside a Vector.
Vector sysvals = svGroup.getSystemValues();

//You will notice that this is one of my system values.
SystemValue mySystemValue = (SystemValue)sysvals.elementAt(0);
System.out.println(mySystemValue.getName()+" - "+mySystemValue.getGroupDescription());

//We can add another SystemValue object from another system into the group.
AS400 sys2 = new AS400("otherSystem.myCompany.com");
SystemValue sv = new SystemValue(sys2, "QDATE");
sysvals.addElement(sv);

//Now refresh the entire group of system values all at once.
//It does not matter if some system values are from different iSeries 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();
```

Trace class

The Trace class allows the Java program to log trace points and diagnostic messages. This information helps reproduce and diagnose problems.

Trace class

Note: You can also set tracing by using the trace system properties.

The Trace class logs the following categories of information:

Information category	Description
Conversion	Logs character set conversions between Unicode and code pages. This category is used only by the IBM Toolbox for Java classes.
Datastream	Logs the data that flows between the iSeries and the Java program. This category is used only by the IBM Toolbox for Java classes.
Diagnostic	Logs state information.

Information category	Description
Error	Logs additional errors that cause an exception.
Information	Traces the flow through a program.
PCML	This category is used to determine how PCML interprets the data that is sent to and from the server.
Proxy	This category is used by IBM Toolbox for Java classes to log data flow between the client and the proxy server.
Warning	Logs information about errors the program was able to recover from.
All	This category is used to enable or disable tracing for all of the above categories at once. Trace information cannot be directly logged to this category.

The IBM Toolbox for Java classes also use the trace categories. When a Java program enables logging, IBM Toolbox for Java information is included with the information that is recorded by the application.

You can enable the trace for a single category or a set of categories. Once the categories are selected, use the setTraceOn method to turn tracing on and off. Data is written to the log using the log method.

You can send trace data for different components to separate logs. Trace data, by default, is written to the default log. Use component tracing to write application-specific trace data to a separate log or standard output. By using component tracing, you can easily separate trace data for a specific application from other data.

Excessive logging can impact performance. Use the isTraceOn method to query the current state of the trace. Your Java program can use this method to determine whether it builds the trace record before it calls the log method. Calling the log method when logging is off is not an error, but it takes more time.

The default is to write log information to standard out. To redirect the log to a file, call the setFileName() method from your Java application. In general, this works only for Java applications because most browsers do not give applets access to write to the local file system.

Logging is off by default. Java programs provide a way for the user to turn on logging so that it is easy to enable logging. For example, the application can parse for a command line parameter that indicates which category of data is logged. The user can set this parameter when log information is needed.

Examples

Note: Read the Code example disclaimer for important legal information.

The following examples show how to use the Trace class.

Example Using setTraceOn() and writing data to a log by using the log method

```
// 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);
```

Example: Using Trace

In the following code, Method 2 is the preferable way to use Trace.

```
// 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);
}
```

Example: Using component tracing

```
// 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");
```

Users and groups

The user and group classes allow you to get a list of users and user groups on the iSeries server as well as information about each user through a Java program.

Note: IBM Toolbox for Java also provides resource classes that present a generic framework and consistent programming interface for working with various iSeries objects and lists. After reading about the classes in the access package and the resource package, you can choose the object that works best for your application. The resource classes for working with users are RUser and RUserList.

Some of the user information you can retrieve includes previous sign-on date, status, date the password was last changed, date the password expires, and user class. When you access the User object, use the setSystem() method to set the system name and the setName() method to set the user name. After those steps, you use the loadUserInformation() method to get the information from the iSeries.

The UserGroup object represents a special user whose user profile is a group profile. Using the getMembers() method, a list of users that are members of the group can be returned.

The Java program can iterate through the list using an enumeration. All elements in the enumeration are User objects; for example:

```
// 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);
}
```

Retrieving information about users and groups

You use a UserList to get a list of the following:

- All users and groups
- Only groups
- All users who are members of groups
- All users who are not members of groups

The only property of the UserList object that must be set is the AS400 object that represents the system from which the list of users is to be retrieved.

By default, all users are returned. Use a combination of setUserInfo() and setGroupInfo() to specify exactly which users are returned.

Example: Using a UserList to list all of the users in a given group.

UserSpace class

The UserSpace class represents a user space on the server. Required parameters are the name of the user space and the AS400 object that represents the server that has the user space. Methods exist in user space class to do the following:

- Create a user space.
- Delete a user space.
- Read from a user space.
- Write to user space.
- Get the attributes of a user space. A Java program can get the initial value, length value, and automatic extendible attributes of a user space.
- Set the attributes of a user space. A Java program can set the initial value, length value, and automatic extendible attributes of a user space.

The UserSpace object requires the integrated file system path name of the program. See integrated file system path names for more information.

Using the UserSpace class causes the AS400 object to connect to the server. See managing connections for information about managing connections.

The following example creates a user space, then writes data to it.

Note: Read the Code example disclaimer for important legal information.

```
                    // 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);
```

Commtrace classes

The IBM Toolbox for Java commtrace classes enable your Java programs to work with communications trace data for a specified LAN (Ethernet or token ring) line description. The commtrace package includes a class that you can run as a standalone utility program to format communications trace data.

When you dump a communications trace for an iSeries server to a stream file, the information is saved in a binary format. The commtrace classes enable you to work with the various components of the stream file.

Note: Communications trace files may contain confidential information, for example, unencrypted passwords. When the communications trace file is on the iSeries server, only users who have *SERVICE special authority can access the trace data. If you move the file to a client, make sure that you protect the file in an appropriate manner. For more information about communications traces, see the links at the bottom of this page.

Use the commtrace classes to perform the following tasks:

- Format the raw trace data.
- Parse any of the data to extract the information you want. You can parse both raw and formatted data, provided that you used the commtrace classes to format the data.

For a visual representation that shows how the commtrace classes represent the structures in a communications trace file, see the following page:

"Commtrace model" on page 174

The commtrace package includes the following classes:

"Format and FormatProperties classes" on page 176: The Format class reads both raw data and formatted data from a communications trace. FormatProperties sets the properties for your Format object, such as start and end times, IP addresses, ports, and so on.

Note: You can also run the Format class as a standalone program.

"Prolog class" on page 179: Retrieves information from the initial 256-byte section of an iSeries server communications trace.

"Frame class" on page 179: Retrieves information about the frames of the communications trace.

"LanHeader class" on page 180: Retrieves information from the section of data that occurs once, near the beginning of a frame. This section typically contains hardware-specific information that includes general information about the frame, such as the frame number, data length, and so on.

"IPPacket class" on page 180: Retrieves information from the data in the packet. IPPacket is the abstract parent class for the different types of data packets that are supported by the commtrace package.

"Header class" on page 181: Retrieves information from the packet header and the associated data. Header is the abstract parent class for the different types of packet headers that are supported by the commtrace package.

Most of the remaining classes in the com.ibm.as400.commtrace package are specific to the type of trace data that you want to work with. For more information about communications traces and about all the commtrace classes, refer to following pages:

Communications trace

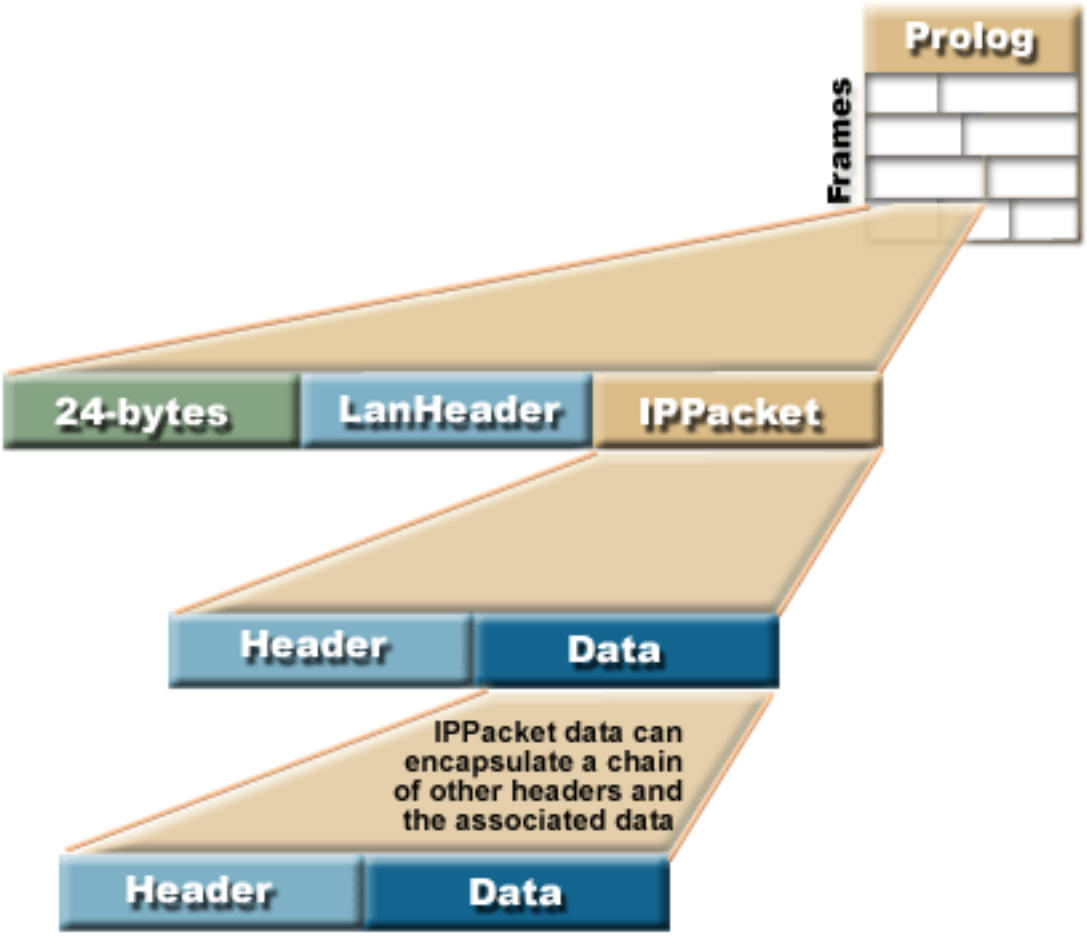
Commtrace model

The following illustration shows how the commtrace classes correspond to a communications trace file. The graphic also indicates the naming conventions that the commtrace classes use for the components in a communications trace.

Prolog: An initial 256-byte section of a communications trace for a LAN line description. The Prolog contains general information about the trace, such as start and end times, number of bytes collected, and so on.

Frame: Records of varying length that contain the data that the iSeries server transmitted during the communcations trace.

Figure 1: Commtrace model



Each Frame in the trace file contains two initial sections (that provide general information about the contents of the frame) and the packet that the iSeries server transmitted between itself and a different point on the network.

The very first 24-byte section of data contains general information about the contents of the frame, such as the frame number and data length. Use the Frame class to process this information.

LanHeader: A section of data that occurs once in each frame (after the initial 24-bytes of data) and contains general information about the IPacket that follows.

IPPacket: A section of data, composed of one ore more headers and the associated data. IPPacket represents all the data packets that the network transmitted for this frame during the communications trace. IPPacket is an abstract class, so you will use the various concrete subclasses to process the headers and data in the packets.

Header: A section of data at the beginning of an IPPacket that describes the following packet data and, when appropriate, points to the next header. In the commtrace package, Header is an abstract class, so you will use the various concrete subclasses to process the data.

Long description of Figure 1: Commtrace model (rzahh587.gif):

found in IBM Toolbox for Java: Commtrace classes

This figure illustrates in a general way how the commtrace classes correspond to a communications trace file.

Description

The figure is composed of the following:

- A square image in the right background labeled 'Prolog,'. This square is divided into rows that represents a section of a communications trace for a LAN line description. Written vertically on the left side of the image is the word 'Frames'
- An image in the left foreground of a horizontal rectangle. This rectangle is connected to the main square in the background by lines on each side. The rectangle represents a frame in the trace file and is divided into 3 sections:
  - A green section labeled 24-bytes that represents thesection of data that contains general information about the contents of the frame.
  - A light blue section labeled LanHeader that indicates the section of data that contains general information about the IPacket that follows..
  - A tan section labeled IPPacket that indicates the section of data composed of one or more headers and the associated data.
- Another horizontal rectangle is below the first. A line on each side of the rectangle connects it to the IPPacket. The rectangle represents a detailed version of the IPPacket section and consists of 2 sections:
  - A light blue section labeled Header that indicates the section of data at the beginning of an IPPacket
  - A dark blue section labeled Data that indicates the packet data and points to the next header.
- A third horizontal rectangle appears below the dark blue Data section. A line on each side of the rectangle connects it to the Data section. The rectangle represents the 2 sections of the IPPacket and states 'IPPacket information encapsulates a chain of other headers and associated data':
-
  - A light blue section labeled Header that indicates the section of data at the beginning of an IPPacket
  - A dark blue section labeled Data that indicates the packet data and points to the next header.

A frame (first horizontal rectangle) in a trace file (background image) contains 2 sections of data, the 24-byte section (green) of data and the LanHeader (light blue), along with the packet (tan).

The IPPacket (second horizontal rectangle) processed in the frame consists of the Header (light blue) and Data (dark blue) which in turn point to another header and data (third horizontal rectangle). The second and third rectangles are joined by connecting lines on each side from the Data section and back to the IPPacket which shows the continuous chain for headers and associated data in a communications trace.

Format and FormatProperties classes

The Format class serves as the interface between the calling program and the frames of the trace. The FormatProperties class enables you to set and retrieve properties that determine how the Format object behaves when it encounters information in the Frames of the trace.

Format class

Use the format class to read both the raw trace data and the trace data that you have already formatted by using the commtrace classes.

Note: You cannot use the commtrace classes to read a communications trace that you formatted by using the Print Communications Trace (PRTCMNTRC) control language command.

Use the Format class to parse and format the information in a trace, then send that formatted information to a file or a print device. Additionally, you might want to create a graphical front end that displays the

information in a standalone application or within a browser. When you want to select only specific data, use the Format class to supply that information to your Java program. For example you could use the Format class to read IP addresses out of a trace and then use that data in your program.

The Format constructors accept arguments that represent unformatted data, such as an IFSFileInputStream object, a local file, or the binary trace file. To display a trace that you have already formatted, use the default Format constructor, then use Format.openIFSFile() or Format.openLclFile() to specify the formatted file that you want to display.

Examples

The following examples show how you can display a saved trace or format a binary trace.

Note: Read the Code example disclaimer for important legal information.

Example: Displaying a saved trace

```
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);
```

Example: Formatting a binary trace

```
// 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();
```

Running Format as a standalone utility

You can also run the Format class as a standalone utility program. For more information, see the following topic:

Running Format as a standalone program

FormatProperties class

Use the FormatProperties class to specify and retrieve the properties for your Format object. In other words, when you use the Format class to send information to a file, use the FormatProperties class to filter the information that you want to send.

These properties specify how you want your Format object to handle the information that it encounters in the Frames of the communications trace. The default behavior is for the Format object to ignore properties for which you have not given a specific value.

The FormatProperties class provides constants that you use to set properties. Setting properties enables the Format object to verify which filters you want to use. For example, the following code sets a Format object to display a progess dialog and not display broadcast frames:

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

Most of the properties are available to your Format object as filters that you set to explicitly include specific data. Once you set the filters, the Format object displays only data that matches those filters. For example, the following code set a filter to display frames that occurred between a particular start and end time:

```
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");
```

Example

The following example shows how you can use many of the commtrace classes, including the Format and FormatProperties classes, to display trace information to your monitor:

"Example: Using the commtrace classes" on page 182

Javadoc reference documentation

For more information about the Format and FormatProperties classes, see the following Javadoc reference documentation:

Format

FormatProperties

Running Format as a standalone program:

In addition to using the Format class in your Java programs, you can run it as a standalone, command line utility to format a communications trace. The program connects an IFSFileOutputStream to the specified outfile and writes the data to that file.

Running format as a standalone utility enables you to format files by using the processing power and storage space of your iSeries server.

Running Format from a command line

To run the Format utility from a command line prompt, use the following command:

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

where [options] equals one or more of the available options. Options include:

- The system to which you want to connect
- The userID and password for the system
- The communications trace that you want to parse
- The file in which you want to store the results

For a complete list of available options, see the following information:

Javadoc reference documentation for the Format class

Running Format remotely

To run this class remotely use the JavaApplicationCall class:

```
// 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
}
```

Prolog class

The Prolog class represents the initial 256-byte section of a communications trace for a LAN line description. The Prolog contains general information about the trace, such as start and end times, number of bytes collected, and so on. Use the Prolog class to retrieve information from this section of trace, which you can then print, display, filter, or process in some other way.

The Prolog class provides methods that allow you to perform a variety of actions that include the following:

- Retrieve values from the fields of the prolog, such as the trace description, Ethernet Type, data direction, IP address, and so on
- Return a formatted String that contains all the fields of the prolog
- Test the prolog fields for invalid data

Example

The following example shows how you can use many of the commtrace classes, including the Prolog class, to display trace information to your monitor:

"Example: Using the commtrace classes" on page 182

Javadoc reference documentation

For more information about the Prolog class, see the following Javadoc reference documentation:

Prolog

Frame class

The Frame class represents all the data in one record, or frame, in a communications trace for a LAN line description. Each Frame contains three main sections of data that appear in the following order:

1. An initial 24-byte section that contains general information about the frame
2. General information about the frame (represented by the LanHeader class)
3. The packet data (represented by subclasses of the IPacket abstract class)

Use the Frame class to parse and create a printable representation the data in the frame. The Frame class maintains the packet data in a linked list-like structure that uses specific formats. For specific information about the possible formats for packet data in a frame and for general information about the structure of a frame, see the following:

Commtrace model

The Frame class provides methods that allow you to perform a variety of actions that include the following:

- Retrieve the data packet
- Retrieve the number, status, and type of the frame
- Return specific data from the frame as a formatted String

You can use the following process to access the data in a packet:

1. Use Frame.getPacket() to retrieve the packet
2. Access the data in the header by calling Packet.getHeader()
3. After you retrieve the header, call Header.getType() to find the type
4. Use the specific Header subclass to to access the data associated with that header (the payload) and any additional headers

Example

The following example shows how you can use many of the commtrace classes, including the Format and FormatProperties classes, to display trace information to your monitor:

"Example: Using the commtrace classes" on page 182

LanHeader class

The LanHeader class represents the section of data in frame that occurs between the initial 24-byte section of information and the packet data. This data contains general information about the frame.

Use the LanHeader class to parse and print the information in the LanHeader. The kind of information contained by the LanHeader includes:

- The byte that identifies the start of the first header in this packet
- Medium Access Control (MAC) addresses
- Token ring addresses and routing information

LanHeader also provides two methods that enable you to return a formatted String that contains the following:

- Token ring routing data
- Source MAC addresses, destination MAC addresses, frame format, and frame type

Javadoc reference documentation

For more information about the LanHeader class, see the following Javadoc reference documentation:

LanHeader

IPPacket class

The IPPacket class is the abstract superclass for creating classes that represent specific kinds of packets. The subclasses of IPPacket include:

- ARPPacket
- IP4Packet
- IP6Packet
- UnknownPacket

Packet classes enable you to retrieve the type of packet and access the raw data (the header and payload) that the packet contains. All the subclasses use similar constructors and include one additional method that returns a printable version of the packet contents as a String.

All the Packet class constructors take a byte array of packet data as an argument, but the ARPPacket also requires an integer that specifies the type of frame. Creating an instance of a Packet class automatically creates the appropriate Header object.

The Packet classes provide methods that allow you to perform a variety of actions that include the following:
- Retrieve the name and type of the packet
- Set the type of the packet
- Return the top-level Header object associated with the packet
- Return all the packet data as an unformatted String
- Return specific data from the packet as a formatted String

Javadoc reference documentation

For more information about the Packet classes, see the following Javadoc reference documentation:

IPPacket

ARPPacket

IP4Packet

IP6Packet

UnknownPacket

Header class

The Header class is the abstract superclass for creating classes that represent specific kinds of packet headers. Packet headers include the associated data (or payload), which can be other headers and payloads. The subclasses of Header include:
- ARPHeader
- ExtHeader
- ICMP4Header
- ICMP6Header
- IP4Header
- IP6Header
- TCPHeader
- UDPHeader
- UnknownHeader

Header classes enable you to retrieve the data for the header and the payload. One header can encapsulate other headers and their payloads.

Creating an instance of a Packet class automatically creates the appropriate Header object. The Header classes provide methods that allow you to perform a variety of actions that include the following:

- Return the length, name, and type of the header
- Retrieve the data in the header as a byte array
- Retrieve the next header in the packet
- Retrieve the payload as a byte array, ASCII string, and hexadecimal string
- Return all the header data as an unformatted String
- Return specific data from the header as a formatted String

Javadoc reference documentation

For more information about the Header classes, see the following Javadoc reference documentation:

Header

ARPHeader

ExtHeader

ICMP4Header

ICMP6Header

IP4Header

IP6Header

TCPHeader

UDPHeader

UnknownHeader

Example: Using the commtrace classes

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////
//
// 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 Classes

IBM Toolbox for Java HTML classes assist you in:

- Setting up forms and tables for HTML pages
- Aligning text
- Working with a variety of HTML tags
- Creating Extensible Stylesheet Language (XSL) formatting object (FO) source data
- Altering the language and direction of text
- Creating ordered and unordered lists
- Creating file lists and HTML hierarchical trees (and the elements within them)
- Adding tag attributes not already defined in the HTML classes (for example, bgcolor and style attributes)

The HTML classes implement the HTMLTagElement interface. Each class produces an HTML tag for a specific element type. The tag may be retrieved using the getTag() method and can then be embedded into any HTML document. The tags you generate with the HTML classes are consistent with the HTML 3.2 specification.

The HTML classes can work with servlet classes to get data from the iSeries server. However, they can also be used alone if you supply the table or form data.

Additionally, you can use the HTMLDocument class to easily create HTML pages or XSL FO source data. You can convert XSL FO data into Portable Document Format (PDF) documents. Using the PDF format enables your documents to retain the same graphical appearance when you print them as when you view them electronically.

The HTML classes make it easier to make HTML forms, tables, and other elements:

- BidiOrdering class allows you to alter the language and direction of text.
- DirFilter class enables you to determine if a File object is a directory.
- HTMLAlign class allows you to align blocks of HTML output.
- HTMLDocument class enables you to more easily create HTML or XSL FO source data.
- HTMLFileFilter class allows you to determine if a File object is a file.
- HTMLForm classes help you make forms more easily than CGI scripting.
- HTMLHead class allows you to create head tags for your HTML pages.
- HTMLHeading class allows you to create heading tags for your HTML pages.
- HTMLHyperlink class helps you create links within your HTML pages.
- HTMLImage class allows you to create image tags for your HTML pages.
- HTMLList classes help you create lists for your HTML pages.
- HTMLMeta class allows you to create meta tags for your HTML pages.
- HTMLParameter class specifies parameters available to the HTMLServlet.
- HTMLServlet class allows you to create a server-side include.
- HTMLTable classes help you make tables for your HTML pages.
- HTMLText class allows you to access the font properties within your HTML pages.
- HTMLTree classes allow you to display an HTML hierarchical tree of HTML elements.
- URLEncoder class encodes delimiters to use in a URL string.
- URLParser class allow you to parse a URL string for the URI, properties, and reference.

Note: The jt400Servlet.jar file includes both the HTML and Servlet classes. You must update your CLASSPATH to point to the jt400Servlet.jar file if you want to use the classes in the com.ibm.as400.util.html package.

BidiOrdering class

The BidiOrdering class represents an HTML tag that alters the language and direction of text. An HTML <BDO> string requires two attributes, one for language and the other for the direction of the text.

The BidiOrdering class allows you to:

- Get and set the language attribute
- Get and set the direction of text

For more information about using the <BDO> HTML tag, see the W3C Web site.

Example: Using BidiOrdering

Note: Read the Code example disclaimer for important legal information.

The following example creates a BidiOrdering object and sets its language and direction:

```
// 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();
```

The print statement produces the following tag:

```
<bdo lang="AR" dir="rtl">
  <b>Some Arabic Text.</b>
</bdo>
```

When you use this tag in an HTML page, browsers that can understand the <BDO> tag display the example like this:

.txeT cibarA emoS

HTMLAlign class

The HTMLAlign class enables you to align sections of your HTML document, instead of just aligning individual items, say paragraphs or headings.

The HTMLAlign class represents the <DIV> tag and its associated align attribute. You can use right, left, or center alignment.

You can use this class to perform a variety of actions that include the following:

- Add or remove items from the list of tags you want to align
- Get and set the alignment
- Get and set the direction of the text interpretation

- Get and set the language of the input element
- Get a String representation of the HTMLAlign object

Example: Creating HTMLAlign objects

Note: Read the Code example disclaimer for important legal information.

The following example creates an unordered list, then creates an HTMLAlign object to align the entire list:

```
// 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);
```

The previous example produces the following tag:

```
<div align="center">
<ul type="disc">
  <li>Centered unordered list</li>
  <li>Another item</li>
</ul>
```

When you use this tag in an HTML page, it looks like this:

- Centered unordered list
- Another item

HTMLDocument class

The HTMLDocument class enables you to more easily use existing IBM Toolbox for Java HTML classes to create either HTML pages or Portable Document Format (PDF) documents. When you create an HTMLDocument, you specify whether it contains HTML tags or Extensible Stylesheet Language (XSL) Formatting Object (FOs) tags:

- When you want to create HTML pages, the HTMLDocument class offers you an easier way to group all the required HTML tags. However, HTML pages do not always look the same when you print them as when you view them in a Web browser.
- When you want to create PDF documents, the HTMLDocument class offers you the ability to create XSL FO source that contains all the information you need to produce a PDF document. PDF documents retain the same graphical appearance when you print them as when you view them electronically.

To use HTMLDocument, you need to include an XML parser and an XSLT processor in your CLASSPATH environment variable. For more information, see the following pages:

"Jar files" on page 11

"XML parser and XSLT processor" on page 393

You can process the resulting HTML or XSL source data the way you want, for example, by displaying the HTML, saving the XSL to a file, or using the streamed data in another part of your Java program.

For more information about creating HTML pages and XSL FO source data, see the following pages:

- "Using HTMLDocument to create HTML data"
- "Using HTMLDocument to create XSL FO data" on page 188
- "Examples: Using HTMLDocument" on page 190

Javadoc reference documentation

For more information about the HTMLDocument class, see the following Javadoc reference documentation:

HTMLDocument

Using HTMLDocument to create HTML data:

An HTMLDocument functions as a wrapper that holds the information necessary to create either HTML or Extensible Stylesheet Language (XSL) Formatting Object (FO) source data. When you want to create HTML pages, the HTMLDocument class offers you an easier way to group all the required HTML tags.

Generating HTML source data

When creating HTML source, HTMLDocument retrieves HTML tags from the HTML objects you have created. You can use either HTMLDocument.getTag() to stream all the elements you have defined or the getTag() for each individual HTML object.

HTMLDocument generates HTML data as you define it in your Java program, so be sure that the resulting HTML is complete and correct.

When you call HTMLDocument.getTag(), the HTMLDocument object performs the following actions:

- Generates the opening <HTML> tag. At the end of the data, it generates the closing </HTML> tag.
- Converts your HTMLHead and HTMLMeta objects into HTML tags.
- Generates the opening <BODY> tag immediately after the <HEAD> tag. At the end of the data, just before the closing </HTML> tag, it generates the closing </BODY> tag.

  Note: If you do not specify a <HEAD> tag, HTMLDocument generates the <BODY> tag after the <HTML> tag.
- Converts your remaining HTML objects into HTML tags as your program directs.

Note: HTMLDocument streams the HTML tags as your Java program directs, so make sure that you call the tags in the proper order.

Examples: Using HTMLDocument

The following example shows how to use HTMLDocument to generate HTML source data (and XSL FO source):

"Example: Using HTMLDocument to generate both HTML source and XSL FO source" on page 193

Javadoc reference documentation

For more information about the HTMLDocument class, see the following Javadoc reference documentation:

HTMLDocument

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Using HTMLDocument to create XSL FO data:

An HTMLDocument functions as a wrapper that holds the information necessary to create either Extensible Stylesheet Language (XSL) Formatting Object (FO) or HTML source data. Generated XSL FO source follows the XSL FO formatting model. The model uses rectangular elements, called areas, to hold the individual content elements, which can be images, text, other XSL FOs, or nothing. The following list describes the four basic types of areas:

- Regions function as the highest level container.
- Block areas represent block level elements, for example, paragraphs or list items.
- Line areas represent a line of text inside of a block.
- Inline areas represent parts of a line, for example, a single character, a footnote, or a mathematical equation.

XSL FO tags created by IBM Toolbox for Java adhere to the XSL standards described the W3C recommendation. For more information about XSL, XSL FOs, and the W3C recommendation, see the following:

Extensible Stylesheet Language (XSL) Version 1.0

Generating XSL FO source data

When creating XSL FO source, HTMLDocument properties represent XSL FO tags that specify page size, orientation, and margins. Additionally, HTMLDocument retrieves from many HTML classes the corresponding XSL FO tags for that content element.

After you use HTMLDocument to generate the XSL FO source, you can use an XSL formatter (for example, the XSLReportWriter class) to place the content elements on the pages of a document.

HTMLDocument generates XSL FO source data in two major sections:

- The first section contains the <fo:root> and <fo:layout-master-set> XSL FO tags that hold general page layout information for page height, page width, and page margins. To specify the values for the layout information, use HTMLDocument set methods to set the values for the associated properties.
- The second section contains the XSL FO <fo:page-sequence> tag that holds the individual content elements. To specify individual content elements, which are instances of HTML classes, retrieve the corresponding XSL FO tag from the HTML object. Make sure that for your content elements you use only HTML classes that have the getFoTag() method.

Note: Trying to retrieve XSL FO tags from HTML classes that do not have the getFoTag() method results in a comment tag.

For more information about the HTML classes that include methods for working with XSL FO tags, see the following Javadoc reference documentation:

"XSL FO-enabled classes"

After you create an instance of HTMLDocument and set the layout properties, retrieve XSL FO tags from HTML objects by using the setUseFO(), getFoTag(), and getTag() methods.

- You can use setUseFO() on either the HTMLDocument or the individual HTML objects. When you use setUseFO(), you can retrieve XSL FO tags by using HTMLDocument.getTag().
- Alternatively, you can use the getFoTag() method on either the HTMLDocument or the individual HTML objects. You might want to use this alternative method when you need to be able to generate both XSL FO and HTML source from the HTMLDocument or the HTML objects.

Example: Using HTMLDocument

After you create XSL FO source data, you need to convert that XSL FO data to a form that your users can view and print. The following examples show how to generate XSL FO source data (and HTML source) and convert the XSL FO source data to a PDF document by using the XSLReportWriter and Context classes:

"Example: Using HTMLDocument to generate both HTML source and XSL FO source" on page 193

"Example: Converting XSL FO source data to a PDF" on page 191

Javadoc reference documentation

For more information about the HTMLDocument class, see the following Javadoc reference documentation:

HTMLDocument

The following disclaimer applies to all of the IBM Toolbox for Java examples:

XSL FO-enabled classes:

Many IBM Toolbox for Java HTML classes feature the following methods, which enable instances of those classes to work with HTMLDocument:

- getFoTag()
- getTag()
- setUseFO()

For more information about the HTMLDocument class and about the HTML classes that include methods for working with XSL FOs, see the following Javadoc reference documentation:

- HTMLDocument
- BidiOrdering
- HTMLAlign
- HTMLHead
- HTMLHeading
- HTMLImage
- HTMLList
- HTMLListItem
- HTMLTable
- HTMLTableCaption
- HTMLTableCell
- HTMLTableHeader
- HTMLTableRow
- HTMLTagElement
- OrderedList
- UnorderedList

Examples: Using HTMLDocument:

The following examples show ways that you can use the HTMLDocument class to generate HTML and Extensible Stylesheet Language (XSL) Formatting Object (FO) source data. Both examples stream the data to....? but you can do this or that with it, too?

Example: Using HTMLDocument to generate both HTML source and XSL FO source

The following examples show how to generate both HTML source data and XSL FO source data at the same time:

"Example: Using HTMLDocument to generate both HTML source and XSL FO source" on page 193

Example: Converting XSL FO source data to a PDF

After you create XSL FO source data, you need to convert that XSL FO data to a form that your users can view and print. The following example shows how to convert a file that contains XSL FO source data to a PDF document by using the XSLReportWriter and Context classes:

"Example: Converting XSL FO source data to a PDF" on page 191

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Converting XSL FO source data to a PDF:

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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 jars are part of the IBM ToolBox for Java, and reside in directory
// /QIBM/ProdData/HTTP/Public/jt400/lib on your server.
//
// 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);
    }
}
```

Example: Using HTMLDocument to generate both HTML source and XSL FO source:

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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 form classes

The HTMLForm class represents an HTML form. This class allows you to:
- Add an element, like a button, hyperlink or HTML table to a form
- Remove an element from a form
- Set other form attributes, such as which method to use to send form contents to the server, the hidden parameter list, or the action URL address

The constructor for the HTMLForm object takes a URL address. This address is referred to as an action URL. It is the location of the application on the server that will process the form input. The action URL can be specified on the constructor or by setting the address using the setURL() method. Form attributes are set using various set methods and retrieved using various get methods.

Any HTML tag element may be added to an HTMLForm object using addElement() and removed using removeElement(). Use the following HTML tag element classes in your HTMLForms:
- FormInput classes: represent input elements for an HTML form
- LayoutFormPanel classes: represent a layout of form elements for an HTML form
- TextAreaFormElement: represents a text area element in an HTML form
- LabelFormElement: represents a label for an HTML form element
- SelectFormElement: represents a select input type for an HTML form
- SelectOption: represents an option for a SelectFormElement object in an HTML form
- RadioFormInputGroup: represents a group of radio input objects which allow a user to select one from a group

Of course, you can add other tag elements to a form, including the following:
- HTMLText
- HTMLHyperlink
- HTMLTable

For more information on using the HTMLForm class to create a form, see this example and the resulting output.

FormInput classes:

The FormInput class allows you to:
- Get and set the name of an input element
- Get and set the size of an input element
- Get and set the initial value of an input element

The FormInput class is extended by the classes in the following list. These classes provide a way to create specific types of form input elements and allow you to get and set various attributes or retrieve the HTML tag for the input element:
- ButtonFormInput: Represents a button element for an HTML form
- FileFormInput: Represents a file input type, for an HTML form
- HiddenFormInput: Represents a hidden input type for an HTML form
- ImageFormInput: Represents an image input type for an HTML form.
- ResetFormInput: Represents a reset button input for an HTML form
- SubmitFormInput: Represents a submit button input for an HTML form

- TextFormInput: Represents a single line of text input for an HTML form where you define the maximum number of characters in a line. For a password input type, you use PasswordFormInput, which extends TextFormInput and represents a password input type for an HTML form
- ToggleFormInput: Represents a toggle input type for an HTML form. The user can set or get the text label and specify whether the toggle should be checked or selected. The toggle input type can be one of two:
  - RadioFormInput: Represents a radio button input type for an HTML form. Radio buttons may be placed in groups with the RadioFormInputGroup class; this creates a group of radio buttons where the user selects only one of the choices presented.
  - CheckboxFormInput: Represents a checkbox input type for an HTML form where the user may select more than one from the choices presented, and where the checkbox is initialized as either checked or unchecked.

ButtonFormInput class:

The ButtonFormInput class represents a button element for an HTML form.

The following example shows you how to create a ButtonFormInput object:

```
ButtonFormInput button = new ButtonFormInput("button1", "Press Me", "test()");
System.out.println(button.getTag());
```

This example produces the following tag:

```
<input type="button" name="button1" value="Press Me" onclick="test()" />
```

FileFormInput class:

The FileFormInput class represents a file input type in an HTML form.

The following code example shows you how to create a new FileFormInput object

```
FileFormInput file = new FileFormInput("myFile");
System.out.println(file.getTag());
```

The above code creates the following output:

```
<input type="file" name="myFile" />
```

HiddenFormInput class:

The HiddenFormInput class represents a hidden input type in an HTML form.

The following code example shows how to create a HiddenFormInput object:

```
HiddenFormInput hidden = new HiddenFormInput("account", "123456");
System.out.println(hidden.getTag()):
```

The previous code generates the following tag:

```
<input type="hidden" name="account" value="123456" />
```

In an HTML page, the HiddenInputType does not display. It sends the information (in this case the account number) back to the server.

ImageFormInput class:

The ImageFormInput class represents an image input type in an HTML form.

You can retrieve and update many of the attributes for the ImageFormInput class by using the methods provided.

- Get or set the source
- Get or set the alignment
- Get or set the height
- Get or set the width

Example: Creating an ImageFormInput object

The following code example shows you how to create an ImageFormInput object:

```
ImageFormInput image = new ImageFormInput("myPicture", "myPicture.gif");
image.setAlignment(HTMLConstants.TOP);
image.setHeight(81);
image.setWidth(100);
```

The above code example generates the following tag:

```
<input type="image" name="MyPicture" src="myPicture.gif" align="top" height="81" width="100" />
```

ResetFormInput class:

The ResetFormInput class represents a reset button input type in an HTML form.

The following code example shows you how to create a ResetFormInput object:

```
ResetFormInput reset = new ResetFormInput();
reset.setValue("Reset");
System.out.println(reset.getTag());
```

The above code example generates the following HTML tag:

```
<input type="reset" value="Reset" />
```

SubmitFormInput class:

The SubmitFormInput class represents a submit button input type in an HTML form.

The following code example shows you how to create a SubmitFormInput object:

```
SubmitFormInput submit = new SubmitFormInput();
submit.setValue("Send");
System.out.println(submit.getTag());
```

The code example above generates the following output:

```
<input type="submit" value="Send" />
```

TextFormInput class:

The TextFormInput class represents a single line text input type in an HTML form. The TextFormInput class provides methods that let you get and set the maximum number of characters a user can enter in the text field.

The following example shows you how to create a new TextFormInput object:

```
TextFormInput text = new TextFormInput("userID");
text.setSize(40);
System.out.println(text.getTag());
```

The code example above generates the following tag:

```
<input type="text" name="userID" size="40" />
```

PasswordFormInput class:

The PasswordFormInput class represents a password input field type in an HTML form.

The following code example shows you how to create a new PasswordFormInput object:

```
PasswordFormInput pwd = new PasswordFormInput("password");
pwd.setSize(12);
System.out.println(pwd.getTag());
```

The code example above generates the following tag:

```
<input type="password" name="password" size="12" />
```

RadioFormInput class:

The RadioFormInput class represents a radio button input type in an HTML form. The radio button may be initialized as selected when constructed.

A set of radio buttons with the same control name make a radio button group. The RadioFormInputGroup class creates radio button groups. Only one radio button within the group may be selected at any time. Also, a specific button may be initialized as selected when the group is constructed.

The following code example shows you how to create a RadioFormInput object:

```
RadioFormInput radio = new RadioFormInput("age", "twentysomething", "Age 20 - 29", true);
System.out.println(radio.getTag());
```

The above code example generates the following tag:

```
<input type="radio" name="age" value="twentysomething" checked="checked" />
```

CheckboxFormInput class:

The CheckboxFormInput class represents a checkbox input type in an HTML form. The user may select more than one of the choices presented as checkboxes within a form.

The following example shows you how to create a new CheckboxFormInput object:

```
CheckboxFormInput checkbox = new CheckboxFormInput("uscitizen", "yes", "textLabel", true);
System.out.println(checkbox.getTag());
```

The code above produces the following output:

```
<input type="checkbox" name="uscitizen" value="yes" checked="checked" /> textLabel
```

LayoutFormPanel class:

The LayoutFormPanel class represents a layout of form elements for an HTML form. You can use the methods provided by the LayoutFormPanel to add and remove elements from a panel or get the number of elements in the layout. You may choose to use one of two layouts:

- "GridLayoutFormPanel": Represents a grid layout of form elements for an HTML form
- "LineLayoutFormPanel class" on page 200: Represents a line layout of form elements for an HTML form

GridLayoutFormPanel:

The GridLayoutFormPanel class represents a grid layout of form elements. You use this layout for an HTML form where you specify the number of columns for the grid.

The following example creates a GridLayoutFormPanel object with two columns:

```
// 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);
```

This example produces the following HTML code:

```
<form action=servletURI method="get">
<table border="0">
<tr>
<td>System:</td>
<td><input type="text" name="System" /></td>
</tr>
<tr>
<td>User:</td>
<td><input type="text" name="User" /></td>
</tr>
<tr>
<td>Password:</td>
<td><input type="password" name="Password" /></td>
</tr>
</table>
<input type="submit" name="logon" value="Logon" />
</form>
```

LineLayoutFormPanel class:

The LineLayoutFormPanel class represents a line layout of form elements for an HTML form. The form elements are arranged in a single row within a panel.

Example: Using LineLayoutFormPanel

This example creates a LineLayoutFormPanel object and adds two form elements.

```
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();
```

The code example above generates the following HTML code:

```
<input type="checkbox" name="confidential" value="yes" checked="checked" /> Confidential
<input type="checkbox" name="mailingList" value="yes" /> Join our mailing list <br/>
```

TextAreaFormElement class:

The TextAreaFormElement class represents a text area element in an HTML form. You specify the size of the text area by setting the number of rows and columns. You can determine the size that a text area element is set for with the getRows() and getColumns() methods.

You set the initial text within the text area with the setText() method. You use the getText() method to see what the initial text has been set to.

The following example shows you how to create a TextAreaFormElement:

```
TextAreaFormElement textArea = new TextAreaFormElement("foo", 3, 40);
textArea.setText("Default TEXTAREA value goes here");
System.out.println(textArea.getTag());
```

The code example above generates the following HTML code:

```
<form>
<textarea name="foo" rows="3" cols="40">
Default TEXTAREA value goes here
</textarea>
</form>
```

LabelFormElement class:

The LabelFormElement class represents a label for an HTML form element. You use the LabelFormElement class to label elements of an HTML form such as a text area or password form input . The label is one line of text that you set using the setLabel() method. This text does not respond to user input and is there to make the form easier for the user to understand.

Example: Using LabelFormElement

The following code example shows you how to create a LabelFormElement object:

```
LabelFormElement label = new LabelFormElement("Account Balance");
System.out.println(label.getTag());
```

This example produces the following output:

```
Account Balance
```

SelectFormElement class:

The SelectFormElement class represents a select input type for an HTML form. You can add and remove various options within the select element.

SelectFormElement has methods available that allow you to view and change attributes of the select element:

- Use setMultiple() to set whether or not the user can select more than one option
- Use getOptionCount() to determine how many elements are in the option layout
- Use setSize() to set the number of options visible within the select element and use getSize() to determine the number of visible options.

The following example creates a SelectFormElement object with three options. The SelectFormElement object named *list*, is highlighted. The first two options added specify the option text, name, and select attributes. The third option added is defined by a SelectOption object.

```
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());
```

The above code example produces the following HTML code:

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

SelectOption class:

The SelectOption class represents an option in an HTML SelectFormElement. You use the option form element in a select form.

Methods are provided that you can use to retrieve and set attributes within a SelectOption. For instance, you can set whether the option defaults to being selected. You can also set the input value it will use when the form is submitted.

The following example creates three SelectOption objects within a select form. Each of the following SelectOption objects are highlighted. They are named *option1, option2* and *option3*. The *option3* object is initially selected.

```
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());
```

The above code example produces the following HTML tag:

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

RadioFormInputGroup class:

The RadioFormInputGroup class represents a group of RadioFormInput objects. A user can select only one of the RadioFormInput objects from a RadioFormInputGroup.

The RadioFormInputGroup class methods allow you to work with various attributes of a group of radio buttons. With these methods, you can:

- Add a radio button
- Remove a radio button
- Get or set the name of the radio group

The following example creates a radio button group:

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

The code example above generates the following HTML code:

```
<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
```

HTMLHead class

The HTMLHead class represents an HTML head tag. The head section of an HTML page features an opening and closing head tag that typically contains other tags. Typically, the head tag contains a title tag and possibly meta tags.

Constructors for HTMLHead enable you to construct a head tag that is empty, that contains a title tag, or that contains a title tag and a meta tag. You can easily add title and meta tags to the empty HTMLHead object.

Methods for the HTMLHead class include setting and getting the page title and the meta tags. Define the contents of the meta tags by using the HTMLMeta class. For more information about the HTMLMeta class, see the following page:

HTMLMeta

The following code shows one way to create an HTMLHead tag:

```
// 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);
```

This is the output of the example HTMLHead tag:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>My main page</title>
</head>
```

Javadoc reference documentation

For more information about using the HTMLHead class, see the following Javadoc reference documentation:

HTMLHead

HTMLHeading class

The HTMLHeading class represents an HTML heading. Each heading can have its own alignment and level from 1 (largest font, most importance) to 6.

Methods for the HTMLHeading class include:

- Get and set the text for the heading

- Get and set the level of the heading
- Get and set the alignment of the heading
- Get and set the direction of the text interpretation
- Get and set the language of the input element
- Get a String representation of the HTMLHeading object

Example: Creating HTMLHeading objects

The following example creates three HTMLHeading objects:

```
// 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);
```

The previous example produces the following tags:

```
<h1 align="left">Heading</h1>
<h2 align="center">Subheading</h2>
<h3 align="right">Item</h3>
```

HTMLHyperlink class

The HTMLHyperlink class represents an HTML hyperlink tag. You use the HTMLHyperlink class to create a link within your HTML page. You can get and set many attributes of hyperlinks with this class, including:

- Get or set the Uniform Resource Identifier for the link
- Get or set the title for the link
- Get or set the target frame for the link

The HTMLHyperlink class can print the full hyperlink with defined properties so that you can use the output in your HTML page.

The following is an example for HTMLHyperlink:

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

The code above produces the following tag:

```
<a href="http://www.ibm.com/as400/toolbox">IBM Toolbox for Java home page</a>
```

HTMLImage class

The HTMLImage class allows you to create image tags for your HTML page. The HTMLImage class provides methods that allow you to get and set image attributes, including the following:

- Get or set the height of the image
- Get or set the width of the image
- Get or set the name of the image
- Get or set the alternate text for the image
- Get or set the horizontal space around the image
- Get or set the vertical space around the image

- Get or set the absolute or relative reference to the image
- Retrieve a String representation of the HTMLImage object

The following example shows one way to create an HTMLImage object:

```
// 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);
```

The print statement produces the following tag on a single line. Text wrapping is for display purposes only.

```
<img src="http://myWebSite/picture.gif"
     alt="Alternate text for this graphic"
     height="94" width="105" />
```

HTMLList classes

The HTMLList classes allow you to easily create lists within your HTML pages. These classes provide methods to get and set various attributes of the lists and the items within the lists.

In particular, the parent class HTMLList provides a method to produce a compact list that displays items in as small a vertical space as possible.

- Methods for HTMLList include:
  - Compact the list
  - Add and remove items from the list
  - Add and remove lists from the list (making it possible to nest lists)
- Methods for HTMLListItem include:
  - Get and set the contents of the item
  - Get and set the direction of the text interpretation
  - Get and set the language of the input element

Use the subclasses of HTMLList and HTMLListItem to create your HTML lists:

- OrderedList and OrderedListItem
- UnorderedList and UnorderedListItem

For coding snippets, see the following examples:

- Example: Creating ordered lists
- Example: Creating unordered lists
- Example: Creating nested lists

OrderedList and OrderedListItem

Use the OrderedList and OrderedListItem classes to create ordered lists in your HTML pages.

- Methods for OrderedList include:
  - Get and set the starting number for the first item in the list
  - Get and set the type (or style) for the item numbers
- Methods for OrderedListItem include:
  - Get and set the number for the item
  - Get and set the type (or style) for the item number

By using the methods in OrderedListItem, you can override the numbering and type for a specific item in the list.

See the example for creating ordered lists.

UnorderedList and UnorderedListItem

Use the UnorderedList and UnorderedListItem classes to create unordered lists in your HTML pages.

- Methods for UnorderedList include:
  - Get and set the type (or style) for the items
- Methods for UnorderedListItem include:
  - Get and set the type (or style) for the item

See the example for creating unordered lists.

Examples: Using HTMLList classes

The following examples show you how to use the HTMLList classes to create ordered lists, unordered lists, and nested lists.

Example: Creating ordered lists

The following example creates an ordered list:

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

The previous example produces the following tags:

```
<ol type="i">
<li>First item</li>
<li>Second item</li>
</ol>
```

Example: Creating unordered lists

The following example creates an unordered list:

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

The previous example produces the following tags:

```
<ul type="square">
<li>First item</li>
<li>Second item</li>
</ul>
```

Example: Creating nested lists

The following example creates a nested list:

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

The previous example produces the following tags:

```
<ol type="i">
<li>First item</li>
<li>Second item</li>
<ul type="square">
<li>First item</li>
<li>Second item</li>
</ul>
<li>Third item</li>
</ol>
```

HTMLMeta class

The HTMLMeta class represents meta-information used within an HTMLHead tag. Attributes in META tags are used when identifying, indexing, and defining information within the HTML document.

Attributes of the META tag include:

- NAME - the name associated with the contents of the META tag
- CONTENT - values associated with the NAME attribute
- HTTP-EQUIV - information gathered by HTTP servers for response message headers
- LANG - the language
- URL - used to redirect users from the current page to another URL

For example, to help search engines determine the contents of a page, you might use the following META tag:

```
<META name="keywords" lang="en-us" content="games, cards, bridge">
```

You can also use HTMLMeta to redirect a user from one page to another.

Methods for the HTMLMeta class include:

- Get and set the NAME attribute
- Get and set the CONTENT attribute
- Get and set the HTTP-EQUIV attribute
- Get and set the LANG attribute
- Get and set the URL attribute

Example: Creating META tags

The following example creates two META tags:

```
// 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);
```

The previous example produces the following tags:

```
<meta name="keywords" content="games, cards, bridge">
<meta http-equiv="Expires" content="Mon, 01 Jun 2000 12:00:00 GMT">
```

HTMLParameter class

The HTMLParameter class represents the parameters you can use with the HTMLServlet class. Each parameter has its own name and value.

Methods for the HTMLParameter class include:

- Get and set the name of the parameter
- Get and set the value of the parameter

Example: Creating HTMLParameter tags

The following example creates an HTMLParameter tag:

```
     // Create an HTMLServletParameter.
HTMLParameter parm = new HTMLParameter ("age", "21");
System.out.println(parm);
```

The previous example produces the following tag:

```
<param name="age" value="21">
```

HTMLServlet class

The HTMLServlet class represents a server-side include. The servlet object specifies the name of the servlet and, optionally, its location. You may also choose to use the default location on the local system.

The HTMLServlet class works with the HTMLParameter class, which specifies the parameters available to the servlet.

Methods for the HTMLServlet class include:

- Add and remove HTMLParameters from the servlet tag
- Get and set the location of the servlet
- Get and set the name of the servlet
- Get and set the alternate text of the servlet

Example: Creating HTMLServlet tags

The following example an HTMLServlet tag:

```
// 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);
```

The previous example produces the following tags:

```
<servlet name="myServlet" codebase="http://server:port/dir">
<param name="parm1" value="value1">
<param name="parm2" value="value2">
The Web server providing this page does not support the SERVLET tag.
</servlet>
```

HTML Table classes

The HTMLTable class allows you to easily set up tables that you can use in HTML pages. This class provides methods to get and set various attributes of the table, including:

- Get and set the width of the border
- Get the number of rows in the table
- Add a column or row to the end of the table
- Remove a column or row at a specified column or row position

The HTMLTable class uses other HTML classes to make creating a table easier. The other HTML classes that work to create tables are:

- HTMLTableCell: Creates a table cell
- HTMLTableRow: Creates a table row
- HTMLTableHeader: Creates a table header cell
- HTMLTableCaption: Creates a table caption

Example: Using HTMLTable classes

The following example shows how to use the HTMLTable classes::

"Example: Using HTMLTable classes" on page 585

HTMLTableCell class:

The HTMLTableCell class takes any HTMLTagElement object as input and creates the table cell tag with the specified element. The element can be set on the constructor or through either of two setElement() methods.

Many cell attributes can be retrieved or updating using methods that are provided in the HTMLTableCell class. Some of the actions you can do with these methods are:

- Get or set the row span
- Get or set the cell height
- Set whether the cell data will use normal HTML line breaking conventions

The following example creates an HTMLTableCell object and displays the tag:

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

The getTag() method above gives the output of the example:

<td align=″center″>IBM Home Page</td>

HTMLTableRow class:

The HTMLTableRow class creates a row within a table. This class provides various methods for getting and setting attributes of a row. Some things you can do with these methods are:

- Add or remove a column from the row
- Get column data at the specified column Index
- Get column index for the column with the specified cell.
- Get the number of columns in a row
- Set horizontal and vertical alignments

The following is an example for HTMLTableRow:

```
// Create a row and set the alignment.
HTMLTableRow row = new HTMLTableRow();
row.setHorizontalAlignment(HTMLTableRow.CENTER);

// Create and add the column information to the row.
HTMLText account = new HTMLText(customers_[rowIndex].getAccount());
HTMLText name = new HTMLText(customers_[rowIndex].getName());
HTMLText balance = new HTMLText(customers_[rowIndex].getBalance());

row.addColumn(new HTMLTableCell(account));
row.addColumn(new HTMLTableCell(name));
row.addColumn(new HTMLTableCell(balance));

// Add the row to an HTMLTable object (assume that the table already exists).
table.addRow(row);
```

HTMLTableHeader class:

The HTMLTableHeader class inherits from the HTMLTableCell class. It creates a specific type of cell, the header cell, giving you a **<th>** cell instead of a **<td>** cell. Like the HTMLTableCell class, you call various methods in order to update or retrieve attributes of the header cell.

The following is an example for HTMLTableHeader:

```
// Create the table headers.
HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("ACCOUNT"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NAME"));
HTMLTableHeader balance_header = new HTMLTableHeader();
HTMLText balance = new HTMLText("BALANCE");
balance_header.setElement(balance);

// Add the table headers to an HTMLTable object (assume that the table already exists).
table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);
```

HTMLTableCaption class:

The HTMLTableCaption class creates a caption for your HTML table. The class provides methods for updating and retrieving the attributes of the caption. For example, you can use the setAlignment() method to specify to which part of the table the caption should be aligned. The following is an example for HTMLTableCaption:

```
// Create a default HTMLTableCaption object and set the caption text.
HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Customer Account Balances - January 1, 2000");

// Add the table caption to an HTMLTable object (assume that the table already exists).
table.setCaption(caption);
```

HTML Text class

The HTMLText class allows you to access text properties for your HTML page. Using the HTMLText class, you can get, set and check the status of many text attributes, including:

- Get or set the size of the font
- Set the bold attribute on (true) or off (false) or determine if it is already on
- Set the underscore attribute on (true) or off (false) or determine if it is already on
- Get or set the horizontal alignment of the text

The following example shows you how to create an HTMLText object and set its bold attribute on and its font size to 5.

```
HTMLText text = new HTMLText("IBM");
text.setBold(true);
text.setSize(5);
System.out.println(text.getTag());
```

The print statement produces the following tag:

```
<font size="5"><b>IBM</b></font>
```

When you use this tag in an HTML page, it looks like this:

IBM

HTMLTree classes

The HTMLTree class allows you to easily set up a hierarchical tree of HTML elements that you can use in HTML pages. This class provides methods to get and set various attributes of the tree, in addition to methods allowing you to:

- Get and set the HTTP Servlet Request
- Add an HTMLTreeElement or FileTreeElement to the tree
- Remove an HTMLTreeElement or FileTreeElement from the tree

The HTMLTree class uses other HTML classes that make it easier to create a hierarchical tree:

- HTMLTreeElement: Creates a tree element
- FileTreeElement: Creates a file tree element
- FileListElement: Creates a file list element
- FileListRenderer: Renders the list of files and directories

Examples: Using HTMLTree classes

The following examples show different ways to use the HTMLTree classes.

- "Example: Using HTMLTree classes" on page 576
- "Example: Creating a traversable integrated file system tree"

Example: Creating a traversable integrated file system tree:

The following example is made of three files that, together, show how you can create a traversable integrated file system tree. The example uses frames to display an HTMLTree and FileListElement in a servlet.

- FileTreeExample.java - generates the HTML frames and starts the servlet
- TreeNav.java - builds and manages the tree
- TreeList.java - displays the contents of selections made in the TreeNav.java class

HTMLTreeElement class:

The HTMLTreeElement class represents an hierarchical element within an HTMLTree or other HTMLTreeElements.

Many tree element attributes can be retrieved or updating using methods that are provided in the HTMLTreeElement class. Some of the actions you can do with these methods are:

- Get or set the visible text of the tree element
- Get or set the URL for the expanded and collapsed icon
- Set whether the tree element will be expanded

The following example creates an HTMLTreeElement object and displays the tag:

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

The getTag() method in the above example generates HTML tags like the following:

```
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>My Web Page</u></font></td>
</tr>
```

```
<tr>
<td> </td>
<td>
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>Another Web Page</u></font> </td>
</tr>
</table>
</td>
</tr>
</table>
```

FileTreeElement class:

The FileTreeElement class represents the Integrated File System within an HTMLTree view.

Many tree element attributes can be retrieved or updating using methods that are provided in the HTMLTreeElement class. You can also get and set the name and path of NetServer shared drives.

Some of the actions these methods enable you to perform are:

- Get or set the URL for the expanded and collapsed icon (inherited method)
- Set whether the tree element will be expanded (inherited method)
- Get or set the name of the NetServer shared drive
- Get or set the path of the NetServer shared drive

Example: Using FileTreeElement

The following example creates a FileTreeElement object and displays the tag:

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

The getTag() method above gives the output of the example.

FileListElement class:

The FileListElement class allows you to create a file list element, which represents the contents of an integrated file system directory.

You can use the FileListElement object to represent the contents of a NetServer shared drive by getting and setting the name and path of NetServer shared drives.

The FileListElement class provides methods that allow you to:
- List and sort the elements of the file list
- Get and set the HTTP Servlet Request
- Get and set the FileListRenderer
- Get and set the HTMLTable with which to display the file list
- Get or set the name of a NetServer shared drive
- Get or set the path of a NetServer shared drive

You can use the FileListElement class with other classes in the html package:
- With a FileListRenderer, you can specify how you want to display the list of files
- With the FileTreeElement class, you can create a traversable list of integrated file system files or NetServer shared files

The FileListElement javadoc shows how to create and display a FileListElement object.

Example: Using FileListElement to create a traversable integrated file system tree

The following example shows how you can use the FileListElement class with HTMLTree classes (FileTreeElement and HTMLTreeElement) to create a traversable integrated file system tree. The example also includes code for setting the path of a NetServer shared drive.

"Example: Creating a traversable integrated file system tree" on page 212

FileListRenderer class:

The FileListRenderer class renders any field for File objects (directories and files) in a FileListElement.

The FileListRenderer class offers methods that allow you to perform the following actions:
- Get the name of the directory
- Get the name of the file
- Get the name of the parent directory
- Return the row data that you want to display in the FileListElement

This example creates an FileListElement object with a renderer:

```
// 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));
```

If you don't want to use the default renderer, you can extend FileListRenderer and override methods or create new ones. For example, you might want to ensure that you prevent passing the names of specific

directories or files with certain extensions to the FileListElement. By extending the class and overriding the appropriate method, you can return null for these files and directories, ensuring that they are not displayed.

To fully customize the rows within a FileListElement, use the getRowData() method. An example of customizing row data using getRowData() might be adding a column to the row data or rearranging the columns. When the default behavior of FileListRenderer is satisfactory, you need no additional programming because the FileListElement class creates a default FileListRenderer.

ReportWriter classes

The com.ibm.as400.util.reportwriter package provides classes that enable you to use your iSeries to more easily access and format data from an XML source file or data produced by servlets or JavaServer Pages(TM). The reportwriter package is a convenient way to name three different but related packages:

- com.ibm.as400.util.reportwriter.pclwriter
- com.ibm.as400.util.reportwriter.pdfwriter
- com.ibm.as400.util.reportwriter.processor

These packages include a variety of classes that allow you to format XML data streams and generate reports in those formats. Make sure you have the necessary jar files in your CLASSPATH, including an XML parser and an XSLT processor. For more information, see the following pages:

Jar files

"XML parser and XSLT processor" on page 393

.

Context classes (in the pclwriter and pdfwriter packages) define methods that the ReportProcessor classes need to render XML and JSP data in the chosen format:

- Use PCLContext in combination with a ReportWriter class to generate a report in the Hewlett Packard Printer Control Language (PCL) format.
- Use PDFContext in combination with a ReportWriter class to generate a report in the Adobe Portable Document Format (PDF).

ReportProcessor classes (in the processor package) enable you to generate formatted reports from information your application gathers from XML source data, Java servlets, and JavaServer Pages (JSPs).

- Use the JSPReportProcessor class to retrieve data from servlets and JSP pages to produce reports in the available formats (contexts).
- Use the XSLReportProcessor class to process your XML data with XSL stylesheets to produce reports in the available formats (contexts).

Context classes

The Context classes support specific data formats that, in combination with the OutputQueue and SpooledFileOutputStream classes, enable the ReportWriter classes to generate reports in that format and put those reports in a spool file.

Your application only has to create an instance of the Context class, which the ReportWriter classes then use to generate the reports. Your application never directly calls any of the methods in either Context class. The PCLContext and PDFContext methods are meant to be used internally by the ReportWriter classes.

Constructing an instance of the Context class requires an OutputStream (from the java.io package) and a PageFormat (from the java.awt.print package). The following examples show how you can construct and use the Context classes with other ReportWriter classes to generate reports:

Example: Using XSLReportProcessor with PCLContext

Example: Using JSPReportProcessor with PDFContext

JSPReportProcessor class

The JSPReportProcessor class enables you to create a document or report from the contents of a JavaServer Page(TM) (JSP) or Java servlet.

Use this class to obtain a JSP or servlet from a given URL and create a document from the contents. The JSP or servlet must provide the document data, including XSL formatting objects. You must specify the output context and the JSP input data source before you can generate any pages of the document. You can then convert the report data to a specified output data stream format.

The JSPReportProcessor class allows you to:

- Process the report
- Set a URL as the template

The following examples show how you can use the JSPReportProcessor and the PDFContext classes to generate a report. The examples include both the Java and the JSP code, which you can view by using the following links. You can also download a ZIP file that contains the example JSP, XML, and XSL source files for both the JSPReportProcessor and XSLReportProcessor examples:

- "Example: Using JSPReportProcessor with PDFContext" on page 597
- "Example: JSPReportProcessor sample JSP file" on page 599

For more information about JSPs, see the Java Server Pages technology Web site.

XSLReportProcessor class

The XSLReportProcessor class enables you to create a document or report by transforming and formatting your XML source data using an XSL stylesheet. Use this class to create the report by using an XSL stylesheet that contains XSL formatting objects (FOs), which must conform to the XSL specification. You then use a Context class to convert the report data to a specified output data stream format.

The XSLReportProcessor class allows you to:

- Set the XSL stylesheet
- Set the XML data source
- Set the XSL FO source
- Process a report

Examples

The following examples show how you can use the XSLReportProcessor and the PCLContext classes to generate a report. The examples include the Java, XML, and XSL code, which you can view by using the following links. You can also download a zip file that contains the example XML, XSL, and JSP source files for both the XSLReportProcessor and JSPReportProcessor examples:

- Example: Using XSLReportProcessor with PCLContext
- Example: XSLReportProcessor sample XML file
- Example: XSLReportProcessor sample XSL file

For more information about XML and XSL, see the XML topic in the Information Center.

Resource classes

The Resource package and its classes have been deprecated. You are advised to use the Access package instead.

The com.ibm.as400.resource package provides a generic framework for working with various AS400 objects and lists. This framework provides a consistent programming interface to all such objects and lists.

The resource package includes the following classes:

- Resource - an object that represents an iSeries resource, such as a user, printer, job, message, or file. Concrete subclasses of resource include:
  - RIFSFile
  - RJavaProgram
  - RJob
  - RPrinter
  - RQueuedMessage
  - RSoftwareResource
  - RUser

  Note: The NetServer classes in the access package are also concrete subclasses of Resource.
- ResourceList - an object that represents a list of iSeries resources, such as a list of users, printers, jobs, messages, or files. Concrete subclasses of resource include:
  - RIFSFileList
  - RJobList
  - RJobLog
  - RMessageQueue
  - RPrinterList
  - RUserList
- Presentation - an object that allows you to present information about resource objects, resource lists, attributes, selections, and sorts to end users

Resource and ChangeableResource classes

The Resource package and its classes have been deprecated. You are advised to use the Access package instead.

The com.ibm.as400.resource.Resource and com.ibm.as400.resource.ChangeableResource abstract classes represent an iSeries resource.

Resource

Resource is an abstract class that provides generic access to the attributes of any resource. Every attribute is identified using an attribute ID, and any given subclass of Resource will normally document the attribute IDs that it supports.

Resource provides only read access to the attribute values.

IBM Toolbox for Java provides the following resource objects:

- RIFSFile - represents a file or directory in the iSeries integrated file system
- RJavaProgram - represents a Java program on the iSeries

- RJob - represents an iSeries job
- RPrinter - represents an iSeries printer
- RQueuedMessage - represents a message in an iSeries message queue or job log
- RSoftwareResource - represents a licensed program on the iSeries
- RUser - represents an iSeries user

ChangeableResource

The ChangeableResource abstract class, a subclass of Resource, adds the ability to change attribute values of an iSeries resource. Attribute changes are cached internally until they are committed or canceled. This allows you to change many attribute values at once.

Note: The NetServer classes in the access package are also concrete subclasses of Resource and ChangeableResource.

Examples

The following examples show how you can directly use concrete subclasses of Resource and ChangeableResource, and also how generic code can work with any Resource or ChangeableResource subclass.

- Retrieving an attribute value from RUser, a concrete subclass of Resource
- Setting attribute values for RJob, a concrete subclass of ChangeableResource
- Using generic code to access resources

Code example disclaimer

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
>
> All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Resource lists

The Resource package and its classes have been deprecated. You are advised to use the Access package instead.

The com.ibm.as400.resource.ResourceList class represents a list of iSeries resources. This is an abstract class which provides generic access to the contents of the list.

IBM Toolbox for Java provides the following resource lists:

- RIFSFileList - represents a list of files and directories in the iSeries integrated file system
- RJobList - represents a list of iSeries jobs
- RJobLog - represents a list of messages in an iSeries job log
- RMessageQueue - represents a list of messages in an iSeries message queue
- RPrinterList - represents a list of iSeries printers
- RUserList - represents a list of iSeries users

A resource list is always either open or closed. The resource list must be open in order to access its contents. In order to provide immediate access to the contents of the list and manage memory efficiently, most resource lists are loaded incrementally.

Resource lists allow you to:

- Open the list
- Close the list
- Access a specific Resource from the list
- Wait for a particular resource to load
- Wait for the complete resource list to load

You can also filter resource lists by using selection values. Every selection value is identified using a selection ID. Similarly, resource lists can be sorted using sort values. Every sort value is identified using a sort ID. Any given subclass of ResourceList will normally document the selection IDs and sort IDs that it supports.

Examples

The following examples show various ways of working with resource lists:

- Example: Getting and printing the contents of a ResourceList
- Example: Using generic code to access a ResourceList
- Example: Presenting a resource list in a servlet (HTML table)

Code example disclaimer

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
>
> All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Presentation class

The Resource package and its classes have been deprecated. You are advised to use the Access package instead.

Every resource object, resource list, and meta data object has an associated com.ibm.as400.resource.Presentation object that provides translated information, such as the name, full name, and icon.

Example: Printing a resource list and its sort values using their Presentations

You can use the Presentation information to present resource objects, resource lists, attributes, selections, and sorts to end users in text format.

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

Security classes

You use the IBM Toolbox for Java security classes to provide secured connections to a server, verify a user's identity, and associate a user with the operating system thread when running on the local server. The security services included are:

- Communications infrastructure using Java Secure Socket Extension (JSSE) provides secure connections both by encrypting the data exchanged between a client and a server session and by performing server authentication.
- Authentication Services provide the ability to:
  - Authenticate a user identity and password against the i5/OS user registry.
  - Ability to assign an identity to the current i5/OS thread.

Secure Sockets Layer

Secure Sockets Layer (SSL) provides secure connections by encrypting the data exchanged between a client and server session, and by performing server authentication.

Using SSL negatively affects performance because SSL connections perform more slowly than connections that do not have encryption. Use SSL connections when the security of the data transferred is a higher priority than performance, for example, when transferring credit card or bank statement information.

Contact your IBM representative for more information.

Using encryption between the IBM Toolbox for Java classes and the i5/OS servers

Using SSL to encrypt data between IBM Toolbox for Java and i5/OS servers:

You can use SSL to encrypt data exchanged between IBM Toolbox for Java classes and i5/OS servers.

| On the client side, use JSSE to encrypt the data. On the server side, you must use the i5/OS digital
| certificate manager to configure the i5/OS servers to exchange encrypted data.

Setting up your client and server to use SSL

To encrypt data flowing between the IBM Toolbox for Java classes and i5/OS servers, complete the following tasks:

1. Set up your servers to exchange encrypted data.
2. Use the SecureAS400 object to force IBM Toolbox for Java to encrypt data.

Note: Completing the first two steps above only creates a secure path between the client and the server. Your application must use the SecureAS400 object to tell the IBM Toolbox for Java which data to encrypt. Data that flows through the SecureAS400 object is the only data that is encrypted. If you use an AS400 object, data is not encrypted and the normal path to the server is used.

Setting up iSeries servers to use SSL:

To set up your iSeries servers to use SSL with IBM Toolbox for Java, complete the following steps.

1. Install the following to your iSeries servers:

   Note: This step is necessary only for server running i5/OS versions prior to V5R4. In V5R4 and subsequent releases, the IBM Cryptographic Access Provider product is no longer available.

   - IBM Cryptographic Access Provider 128-bit for iSeries, 5722-AC3, which provides server-side encryption.
2. Get and configure the server certificate.
3. Apply the certificate to the following iSeries servers that are used by IBM Toolbox for Java:
   - 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

Getting and configuring server certificates

Before you get and configure your server certificate, you need to install the following products:

- IBM HTTP Server for iSeries  (5722-DG1) licensed program
- Base operating system option 34 (Digital Certificate Manager)

The process you follow to get and configure your server certificate depends on the kind of certificate you use:

- If you get a certificate from a trusted authority (such as VeriSign, Inc., or RSA Data Security, Inc.), install the certificate on iSeries then apply it to the host servers.
- If you choose not to use a certificate from a trusted authority, you can build your own certificate to be used on iSeries. Build the certificate by using Digital Certificate Manager:
  1. Create the certificate authority on the iSeries server. See the Information Center topic, Acting as your own CA.
  2. Create a system certificate from the certificate authority that you created.
  3. Assign which host servers will use the system certificate that you created.

Authentication services

Classes are provided by the IBM Toolbox for Java that interact with the security services provided by i5/OS. Specifically, support is provided to authenticate a user identity, sometimes referred to as a *principal*, and password against the i5/OS user registry. A credential representing the authenticated user can then be established. You can use the credential to alter the identity of the current i5/OS thread to perform work under the authorities and permissions of the authenticated user. In effect, this swap of identity results in the thread acting as if a signon was performed by the authenticated user.

Note: The services to establish and swap credentials are only supported for servers at release V5R1M0 or greater.

Overview of support provided

The AS400 object provides authentication for a given user profile and password against the server. You can also retrieve Kerberos tickets and profile tokens that represent authenticated user profiles and passwords for the system.

Note: Using Kerberos tickets requires that you install J2SDK, v1.4 and configure the Java General Security Services (JGSS) Application Programming Interface. For more information about JGSS, see the J2SDK, v1.4 Security Documentation  .

To use Kerberos tickets, set only the system name (and not the password) into the AS400 object. The user identity is retrieved through the JGSS framework. You can set only one means of authentication in an AS400 object at a time. Setting the password clears any Kerberos ticket or profile token.

To use profile tokens, use the getProfileToken() methods to retrieve instances of the ProfileTokenCredential class. Think of profile tokens as a representation of an authenticated user profile and password for a specific server. Profile tokens expire based on time, up to one hour, but can be refreshed in certain cases to provide an extended life span.

Note: If you use the ProfileTokenCredential class, make sure to review the information at the bottom of this page that discuss the methods for setting tokens.

The following example creates a system object and uses that object to generate a profile token. The example then uses the profile token to create another system object, and uses the second system object to connect to the command service:

```
AS400 system = new AS400("mySystemName", "MYUSERID", "MYPASSWORD");
ProfileTokenCredential myPT = system.getProfileToken();
AS400 system2 = new AS400("mySystemName", myPT);
system2.connectService(AS400.COMMAND);
```

Setting thread identities

You can establish a credential on either a remote or local context. Once created, you can serialize or distribute the credential as required by the calling application. When passed to a running process on the associated server, a credential can be used to modify or *swap* the i5/OS thread identity and perform work on behalf of the previously authenticated user.

A practical application of this support might be in a two tier application, with authentication of a user profile and password being performed by a graphical user interface on the first tier (i.e. a PC) and work being performed for that user on the second tier (the server). By utilizing ProfileTokenCredentials, the application can avoid directly passing user IDs and passwords over the network. The profile token can then be distributed to the program on the second tier, which can perform the *swap()* and operate under the i5/OS authorities and permissions assigned to the user.

Note: While inherently more secure than passing a user profile and password due to limited life span, profile tokens should still be considered sensitive information by the application and handled accordingly. Since the token represents an authenticated user and password, it could potentially be exploited by a hostile application to perform work on behalf of that user. It is ultimately the responsibility of the application to ensure that credentials are accessed in a secure manner.

Methods for setting tokens in ProfileTokenCredential

The methods for setting tokens in ProfileTokenCredential class require that you distinguish different ways to specify passwords:

- As a special value, such as *NOPWD or *NOPWDCHK, by using a defined special value integer

- As the password for the user profile by using a String that represents the password

Note: In V5R3, IBM Toolbox for Java deprecates the setToken methods that do not require you to distinguish how to specify the password.

Additionally, the setToken methods allow remote users to specify password special values and allow longer user profile passwords of up to 128 characters.

To specify a password special value integer, such as *NOPWD or *NOPWDCHK, use one of the following methods:

- setToken(AS400Principal principal, int passwordSpecialValue)
- setToken(String name, int passwordSpecialValue)

The ProfileTokenCredential class includes the following static constants for password special value integers:

- ProfileTokenCredential.PW_NOPWD: indicates *NOPWD
- ProfileTokenCredential.PW_NOPWDCHK: indicates *NOPWDCHK

To specify a user profile password as a String, use one of the following methods:

- setTokenExtended(AS400Principal principal, String password)
- setTokenExtended(String name, String password)

The setTokenExended methods do not allow you to pass password special value strings as the password parameter. For example, these methods do not allow a password string of *NOPWD.

For more information, see the following Javadoc reference information:

ProfileTokenCredential

Example

Refer to this code for an example of how to use a profile token credential to swap the i5/OS thread identity and perform work on behalf of a specific user.

Servlet classes

The servlet classes that are provided with IBM Toolbox for Java work with the access classes, which are located on the webserver, to give you access to information located on the iSeries server. You decide how to use the servlet classes to assist you with your own servlet projects.

The following diagram shows how the servlet classes work between the browser, webserver, and iSeries data. A browser connects to the webserver that is running the servlet. jt400Servlet.jar and jt400.jar files reside on the webserver because the servlet classes use some of the access classes to retrieve the data and the HTML classes to present the data. The webserver is connected to the iSeries server where the data is.

Figure 1: How servlets work

"Long description of Figure 1: How servlets work (rzahh585.gif)"

There are four types of servlet classes included with IBM Toolbox for Java:

- Authentication classes
- RowData classes
- RowMetaData classes
- Converter classes

Note: The jt400Servlet.jar file includes both the HTML and Servlet classes. You must update your CLASSPATH to point to both jt400Servlet.jar and jt400.jar if you want to use classes in the com.ibm.as400.util.html and com.ibm.as400.util.servlet packages.

For more information about servlets in general, see the reference section.

Long description of Figure 1: How servlets work (rzahh585.gif)

found in IBM Toolbox for Java: Servlet classes

This figure illustrates in a general way how servlets work.

Description

The figure is composed of the following:

- An image on the left of a personal computer, labeled 'Browser,' that represents an instance of a browser running on a personal computer.
- An image of an iSeries server on the right, labeled 'iSeries Data,' that represents the location of the data that you want the servlet to access.
- An image of an iSeries server in the middle (between the other two images), labeled 'WebServer,' that represents the Web server. Several labeled shapes on the WebServer image indicate files or functions that reside on the WebServer:
  - A green oval labeled Servlet that represents the location of the servlet code.
  - A tan circle labeled jt400Servlet that indicates the location of the jt400Servlet.jar file.
  - A tan circle labeled jt400 that indicates the location of the jt400.jar file.

  Note: The WebServer does not have to be on an iSeries server, but it can be, and can even be the same server as that indicated by the iSeries Data image.
- Lines that connect the images together.

A line labeled HTML connects the Browser (the left image) to a Servlet (the green oval) on the WebServer (middle image). The line is labeled HTML because servlets most often use HTML to 'serve' data to the browser.

The WebServer is running two IBM Toolbox for Java jar files (the tan circles), jt400Servlet.jar and jt400.jar. The classes in jt400Servlet.jar, along with the classes in jt400.jar, enable the WebServer to run a servlet that easily connects to servers that contain iSeries Data (the right image). The line with arrowheads on both ends that connects the two images indicates this connection.

Authentication classes

Two classes in the servlet package perform authentication for servlets: AuthenticationServlet and AS400Servlet.

AuthenticationServlet class

AuthenticationServlet is an HttpServlet implementation that performs basic authentication for servlets. Subclasses of AuthenticationServlet override one or more of the following methods:

- Override the validateAuthority() method to perform the authentication (required)
- Override the bypassAuthentication() method so that the subclass authenticates only certain requests
- Override the postValidation() method to allow additional processing of the request after authentication

The AuthenticationServlet class provides methods that allow you to:

- Initialize the servlet
- Get the authenticated user ID
- Set a user ID after bypassing authentication
- Log exceptions and messages

AS400Servlet class

The AS400Servlet class is an abstract subclass of AuthenticationServlet that represents an HTML servlet. You can use a connection pool to share connections and manage the number of connections to the server that a servlet user can have.

The AS400Servlet class provides methods that allow you to:

- Validate user authority (by overriding the validateAuthority() method of the AuthenticationServlet class)
- Connect to a system
- Get and return connection pool objects to and from the pool
- Close a connection pool
- Get and set the HTML document head tags
- Get and set the HTML document end tags

For more information about servlets in general, see the reference section.

RowData class

The RowData class is an abstract class that provides a way to describe and access a list of data.

There are four main classes that extend the RowData class:

- ListRowData
- RecordListRowData
- ResourceListRowData
- SQLResultSetRowData

The RowData classes allow you to:

- Get and set the current position
- Get the row data at a given column using the getObject() method
- Get the meta data for the row
- Get or set the properties for an object at a given column
- Get the number of rows in the list using the length() method.

RowData position

There are several methods that allow you to get and set the current position within a list. The following table lists both the set and get methods for the RowData classes.

Set methods		**Get methods**
absolute()	next()	getCurrentPosition()
afterLast()	previous()	isAfterLast()
beforeFirst()	relative()	isBeforeFirst()
first()		isFirst()
last()		isLast()

ListRowData class:

The ListRowData class allows you to do the following:
- Add and remove rows to and from the result list.
- Get and set the row
- Get information about the list's columns with the getMetaData() method
- Set column information with the setMetaData() method

The ListRowData class represents a list of data. ListRowData can represent many types of information, including the following, through IBM Toolbox for Java "Access classes" on page 20:
- A directory in the integrated file system
- A list of jobs
- A list of messages in a message queue
- A list of users
- A list of printers
- A list of spooled files

Example

The following example shows how the ListRowData and HTMLTableConverter classes work. The example shows the Java code, HTML code, and HTML look and feel.

"Example: Using ListRowData" on page 619

RecordListRowData class:

The RecordListRowData class allows you to do the following:
- Add and remove rows to and from the record list.
- Get and set the row
- Set the record format with the setRecordFormat method
- Get the record format.

The RecordListRowData class represents a list of records. A record can be obtained from the server in different formats, including:

- A record to be written to or read from a server file
- An entry in a data queue
- The parameter data from a program call
- Any data returned that needs to be converted between the server format and Java format

This example shows you how RecordListRowData and HTMLTableConverter work. It shows the java code, HTML code, and HTML look and feel.

ResourceListRowData class:

The ResourceListRowData class represents a resource list of data. Use ResourceListRowData objects to represent any implementation of the ResourceList interface.

Resource lists are formatted into a series of rows, where each row contains a finite number of columns determined by the number of column attribute IDs. Each column within a row contains an individual data item.

The ResourceListRowData class offers methods that enable you to perform the following actions:

- Get and set column attribute IDs
- Get and set the resource list
- Retrieve the number of rows in the list
- Get the column data for the current row
- Get the property list of the data object
- Get the metadata for the list

Example: Presenting a resource list in a servlet

Code example disclaimer

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
>
> All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

SQLResultSetRowData class:

The SQLResultSetRowData class represents an SQL result set as a list of data. This data is generated by an SQL statement through JDBC. With methods provided, you can get and set the result set metadata.

This example shows you how ListRowData and HTMLTableConverter work. It shows the java code, HTML code, and HTML look and feel.

RowMetaData classes

The RowMetaData class defines an interface that you use to find out information about the columns of a RowData object.

With the RowMetaData classes you can do the following:

- Get the number of columns
- Get the name, type, or size of the column
- Get or set the column label
- Get the precision or scale of the column data
- Determine if the column data is text data

There are three main classes that implement the RowMetaData class. These classes provide all the RowMetaData functions listed above in addition to having their own specific functions:

- ListMetaData
- RecordFormatMetaData
- SQLResultSetMetaData

ListMetaData class:

The ListMetaData lets you get information about and change settings for the columns in a "ListRowData class" on page 226. It uses the setColumns() method to set the number of columns, clearing any previous column information. Alternatively, you can also pass the number of columns when you set the constructor's parameters.

Example

The following example shows how ListMetaData, ListRowData and HTMLTableConverter work. It shows the Java code, HTML code, and HTML look and feel.

"Example: Using ListRowData" on page 619

RecordFormatMetaData class:

The RecordFormatMetaData makes use of the IBM Toolbox for Java RecordFormat class. It allows you to provide the record format when you set the constructor's parameters or use the get and set methods to access the record format.

The following example shows you how to create a RecordFormatMetaData object:

```
// Create a RecordFormatMetaData object from a sequential file's record format.
RecordFormat recordFormat = sequentialFile.getRecordFormat();
RecordFormatMetaData metadata = new RecordFormatMetaData(recordFormat);

// Display the file's column names.
int numberOfColumns = metadata.getColumnCount();
for (int column=0; column < numberOfColumns; column++)
{
   System.out.println(metadata.getColumnName(column));
}
```

SQLResultSetMetaData class:

The SQLResultSetMetaData class returns information about the columns of an SQLResultSetRowData object. You can either provide the result set when you set the constructor's parameters or use the get and set methods to access the result set meta data.

The following example shows you how to create an SQLResultSetMetaData object:

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

Converter classes

You use the converter classes to convert row data into formatted string arrays. The result is in HTML format and ready for presentation on your HTML page. The following classes take care of the conversion for you:

- StringConverter
- HTMLFormConverter
- HTMLTableConverter

StringConverter class:

The StringConverter class is an abstract class that represents a row data string converter. It provides a convert() method to convert row data. This returns a string array representation of that row's data.

HTMLFormConverter class:

The HTMLFormConverter classes extends StringConverter by providing an additional convert method called convertToForms(). This method converts row data into an array of single-row HTML tables. You can use these table tags to display the formatted information on a browser.

You can tailor the appearance of the HTML form by using the various get and set methods to view or change the attributes of the form. For example, some of the attributes that you can set include:

- Alignment
- Cell spacing
- Header hyperlinks
- Width

Example: Using HTMLFormConverter

The following example illustrates using HTMLFormConverter. (You can compile and run this example with a webserver running.)

Using HTMLFormConverter

HTMLTableConverter class:

The HTMLTableConverter class extends StringConverter by providing a convertToTables() method. This method converts row data into an array of HTML tables that a servlet can use to display the list on a browser.

You can use the getTable() and setTable() methods to choose a default table that will be used during conversion. You can set table headers within the HTML table object or you can use the meta data for the header information by setting setUseMetaData() to true.

The setMaximumTableSize() method allows you to limit the number of rows in a single table. If the row data does not all fit within the specified size of table, the converter will produce another HTML table object in the output array. This will continue until all row data has been converted.

Examples

The following examples illustrate how to use the HTMLTableConverter class:

- Example: Using ListRowData
- Example: Using RecordListRowData
- Example: Using SQLResultSetRowData
- Example: Presenting ResourceList in a servlet

Code example disclaimer

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
>
> All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Utility classes

Utility classes help you perform specific tasks.

IBM Toolbox for Java offers the following utilities:

- AS400ToolboxJarMaker: Generates a faster loading IBM Toolbox for Java JAR file by creating a smaller JAR file from a larger one, or by selectively unzipping a JAR file to gain access to the individual content files.
- CommandHelpRetriever: Retrieves and generates help text for i5/OS control language (CL) commands.
- CommandPrompter: Prompts for the parameter on a given command. CommandPrompter offers functionality that is similar to the iSeries CL command prompt (pressing F4) and the same as the Management Central command prompt.
- RunJavaApplication and VRunJavaApplication: Allow you to run a Java program on an iSeries server from a command line prompt.
- JPing: Allows you to query a server to find out which services are active. You can also specify if you want to ping the SSL ports.

Client installation and update classes

The IBM Toolbox for Java classes can be referenced at their location in the integrated file system on the server.

Because program temporary fixes (PTFs) are applied to this location, Java programs that access these classes directly on the server automatically receive these updates. But, Accessing the classes from the server does not always work, specifically for the following situations:

- If a low-speed communication link connects server and the client, the performance of loading the classes from the server may be unacceptable.
- If Java applications use the CLASSPATH environment variable to access the classes on the client file system, you need iSeries Access for Windows to redirect file system calls to the server. It may not be possible for iSeries Access for Windows to reside on the client.

In these cases, installing the classes on the client is a better solution.

AS400ToolboxJarMaker

While the JAR file format was designed to speed up the downloading of Java program files, the AS400ToolboxJarMaker generates an even faster loading IBM Toolbox for Java JAR file through its ability to create a smaller JAR file from a larger one.

AS400ToolboxJarMaker class

Also, the AS400ToolboxJarMaker class can unzip a JAR file for you to gain access to the individual content files for basic use.

Flexibility of AS400ToolboxJarMaker

All of the AS400ToolboxJarMaker functions are performed with the JarMaker class and the AS400ToolboxJarMaker subclass:

- The generic JarMaker tool operates on any JAR or Zip file; it splits a jar file or reduces the size of a jar file by removing classes that are not used.
- The AS400ToolboxJarMaker customizes and extends JarMaker functions for easier use with IBM Toolbox for Java JAR files.

According to your needs, you can invoke the AS400ToolboxJarMaker methods from within your own Java program or from a command line. Call AS400ToolboxJarMaker from the command line by using the following syntax:

```
java utilities.JarMaker [options]
```

where

- options = one or more of the available options

For a complete set of options available to run at a command line prompt, see the following:

- Options for the JarMaker base class
- Extended options for the AS00ToolboxJarMaker subclass

Using AS400ToolboxJarMaker

You can use AS400ToolboxJarMaker to work with JAR files in several ways:

- Uncompress one file bundled within a JAR file
- Split a large JAR file into smaller JAR files
- Exclude any IBM Toolbox for Java files that your application does not need to run

Uncompressing a JAR file

Suppose you wanted to uncompress just one file bundled within a JAR file. AS400ToolboxJarMaker allows you to expand the file into one of the following:

- Current directory (extract(jarFile))
- Another directory (extract(jarFile, outputDirectory))

For example, with the following code, you are extracting AS400.class and all of its dependent classes from jt400.jar:

```
java utilities.AS400ToolboxJarMaker -source jt400.jar
    -extract outputDir
    -requiredFile com/ibm/as400/access/AS400.class
```

Splitting up a single JAR file into multiple, smaller JAR files

Suppose you wanted to split up a large JAR file into smaller JAR files, according to your preference for maximum JAR file size. AS400ToolboxJarMaker, accordingly, provides you with the split(jarFile, splitSize) function.

In the following code, jt400.jar is split into a set of smaller JAR files, none larger than 300KB:

```
java utilities.AS400ToolboxJarMaker -split 300
```

Removing unused files from a JAR file

With AS400ToolboxJarMaker, you can exclude any IBM Toolbox for Java files not needed by your application by selecting only the IBM Toolbox for Java components, languages, and CCSIDs that you need to make your application run. AS400ToolboxJarMaker also provides you with the option of including or excluding the JavaBean files associated with the components you select.

For example, the following command creates a JAR file that contains only those IBM Toolbox for Java classes needed to make the CommandCall and ProgramCall components of the IBM Toolbox for Java work:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall
```

Additionally, if it is unnecessary to convert text strings between Unicode and the double byte character set (DBCS) conversion tables, you can create a 400KB byte smaller JAR file by omitting the unneeded conversion tables with the -ccsid option:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall -ccsid 61952
```

Note: Conversion classes are not included with the program call classes. When including program call classes, you must also explicitly include the conversion classes used by your program by using the -ccsid option.

Components supported by IBM Toolbox for Java:

Following table lists the component IDs that you can specify when invoking the AS400ToolboxJarMaker tool.

- The Component column lists the common name for the component.
- The Keyword column lists the keyword that you should specify when using the `-component` option tag.
- The Constant column lists the Integer value that you should specify in setComponents() and getComponents().

Component	Keyword	Constant
Server object	AS400	AS400ToolboxJarMaker.AS400

Component	Keyword	Constant
Command Call	CommandCall	AS400ToolboxJarMaker.COMMAND_CALL
Connection Pool	ConnectionPool	AS400ToolboxJarMaker.CONNECTION_POOL
Data Areas	DataArea	AS400ToolboxJarMaker.DATA_AREA
Data Description and Conversion	DataDescription	AS400ToolboxJarMaker.DATA_DESCRIPTION
Data Queues	DataQueue	AS400ToolboxJarMaker.DATA_QUEUE
Digital Certificates	DigitalCertificate	AS400ToolboxJarMaker.DIGITAL_CERTIFICATE
FTP	FTP	AS400ToolboxJarMaker.FTP
Integrated File System	IntegratedFileSystem	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM
JAAS	JAAS	AS400ToolboxJarMaker.JAAS
Java Application Call	JavaApplicationCall	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL
JDBC	JDBC	AS400ToolboxJarMaker.JDBC
Jobs and Job Queues	Job	AS400ToolboxJarMaker.JOB
Messages and Message Queues	Message	AS400ToolboxJarMaker.MESSAGE
Numeric Data Types	NumericDataTypes	AS400ToolboxJarMaker.NUMERIC_DATA_TYPES
NetServer	NetServer	AS400ToolboxJarMaker.NETSERVER
Network Print	Print	AS400ToolboxJarMaker.PRINT
Program Call	ProgramCall	AS400ToolboxJarMaker.PROGRAM_CALL
Record Level Access	RecordLevelAccess	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS
Secure server	SecureAS400	AS400ToolboxJarMaker.SECURE_AS400
Service Program Call	ServiceProgramCall	AS400ToolboxJarMaker.SERVICE_PROGRAM_CALL
System Status	SystemStatus	AS400ToolboxJarMaker.SYSTEM_STATUS
System Values	SystemValue	AS400ToolboxJarMaker.SYSTEM_VALUE
Trace and Logging	Trace	AS400ToolboxJarMaker.TRACE
Users and Groups	User	AS400ToolboxJarMaker.USER
User Spaces	UserSpace	AS400ToolboxJarMaker.USER_SPACE
Visual server object	AS400Visual	AS400ToolboxJarMaker.AS400_VISUAL
Visual Command Call	CommandCallVisual	AS400ToolboxJarMaker.COMMAND_CALL_VISUAL
Visual Data Queues	DataQueueVisual	AS400ToolboxJarMaker.DATA_QUEUE_VISUAL
Visual Integrated File System	IntegratedFileSystemVisual	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM_VISUAL
Visual Java Application Call	JavaApplicationCallVisual	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL_VISUAL
Visual JDBC	JDBCVisual	AS400ToolboxJarMaker.JDBC_VISUAL

Component	Keyword	Constant
Visual Jobs and Job Queues	JobVisual	AS400ToolboxJarMaker.JOB_VISUAL
Visual Messages and Message Queues	MessageVisual	AS400ToolboxJarMaker.MESSAGE_VISUAL
Visual Network Print	PrintVisual	AS400ToolboxJarMaker.PRINT_VISUAL
Visual Program Call	ProgramCallVisual	AS400ToolboxJarMaker.PROGRAM_CALL_VISUAL
Visual Record Level Access	RecordLevelAccessVisual	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS_VISUAL
Visual Users and Groups	UserVisual	AS400ToolboxJarMaker.USER_VISUAL

CCSID and Encoding values supported by IBM Toolbox for Java:

The IBM Toolbox for Java is shipped with a set of conversion tables, named according to the CCSID. These tables are used internally by IBM Toolbox for Java classes (such as CharConverter) when converting data that is transferred to or from an iSeries server. For example, the conversion table for CCSID 1027 is in file com/ibm/as400/access/ConvTable1027.class. Conversion tables for the following CCSIDs are included in the IBM Toolbox for Java jar file; other encodings are supported by using the JDK. The central server on the server is no longer used to download tables at runtime. Any specified CCSID for which a conversion table or a JDK encoding cannot be found will cause an exception to be thrown. Some of these tables may be redundant to tables included in your JDK. IBM Toolbox for Java presently supports the following 122 different i5/OS CCSIDs.

For additional information about CCSIDs, including a complete list of CCSIDs that are recognized by iSeries servers, see Globalization.

Supported CCSIDs in IBM Toolbox for Java

CCSID	Format	Description
37	Single-byte EBCDIC	United States and others
273	Single-byte EBCDIC	Austria, Germany
277	Single-byte EBCDIC	Denmark, Norway
278	Single-byte EBCDIC	Finland, Sweden
280	Single-byte EBCDIC	Italy
284	Single-byte EBCDIC	Spain, Latin America
285	Single-byte EBCDIC	United Kingdom
290	Single-byte EBCDIC	Japanese Katakana (single-byte only)
297	Single-byte EBCDIC	France
300	Double-byte EBCDIC	Japanese Graphic (subset of 16684)
367	ASCII/ISO/Windows	ASCII (ANSI X3.4 standard)
420	Single-byte EBCDIC (bidirectional)	Arabic EBCDIC ST4
423	Single-byte EBCDIC	Greek (for compatibility; see 875)
424	Single-byte EBCDIC (bidirectional)	Hebrew EBCDIC ST4
437	ASCII/ISO/Windows	ASCII (USA PC Data)

CCSID	Format	Description
500	Single-byte EBCDIC	Latin-1 (MNCS)
720	ASCII/ISO/Windows	Arabic (MS-DOS)
737	ASCII/ISO/Windows	Greek (MS-DOS)
775	ASCII/ISO/Windows	Baltic (MS-DOS)
813	ASCII/ISO/Windows	ISO 8859-7 (Greek/Latin)
819	ASCII/ISO/Windows	ISO 8859-1 (Latin-1)
833	Single-byte EBCDIC	Korean (single-byte only)
834	Double-byte EBCDIC	Korean Graphic (subset of 4930)
835	Double-byte EBCDIC	Traditional Chinese Graphic
836	Single-byte EBCDIC	Simplified Chinese (single-byte only)
837	Double-byte EBCDIC	Simplified Chinese Graphic
838	Single-byte EBCDIC	Thai
850	ASCII/ISO/Windows	Latin-1
851	ASCII/ISO/Windows	Greek
852	ASCII/ISO/Windows	Latin-2
855	ASCII/ISO/Windows	Cyrillic
857	ASCII/ISO/Windows	Turkish
860	ASCII/ISO/Windows	Portuguese
861	ASCII/ISO/Windows	Iceland
862	ASCII/ISO/Windows (bidirectional)	Hebrew ASCII ST4
863	ASCII/ISO/Windows	Canada
864	ASCII/ISO/Windows (bidirectional)	Arabic ASCII ST5
865	ASCII/ISO/Windows	Denmark/Norway
866	ASCII/ISO/Windows	Cyrillic/Russian
869	ASCII/ISO/Windows	Greek
870	Single-byte EBCDIC	Latin-2
871	Single-byte EBCDIC	Iceland
874	ASCII/ISO/Windows	Thai (subset of 9066)
875	Single-byte EBCDIC	Greek
878	ASCII/ISO/Windows	Russian
880	Single-byte EBCDIC	Cyrillic Multilingual (for compatibility; see 1025)
912	ASCII/ISO/Windows	ISO 8859-2 (Latin-2)
914	ASCII/ISO/Windows	ISO 8859-4 (Latin-4)
915	ASCII/ISO/Windows	ISO 8859-5 (Cyrillic 8-bit)
916	ASCII/ISO/Windows (bidirectional)	ISO 8859-8 (Hebrew) ST5
920	ASCII/ISO/Windows	ISO 8859-9 (Latin-5)
921	ASCII/ISO/Windows	ISO 8859-13 (Baltic 8-bit)
922	ASCII/ISO/Windows	Estonia ISO-8
923	ASCII/ISO/Windows	ISO 8859-15 (Latin-9)
930	Mixed-byte EBCDIC	Japanese (subset of 5026)

CCSID	Format	Description
933	Mixed-byte EBCDIC	Korean (subset of 1364)
935	Mixed-byte EBCDIC	Simplified Chinese (subset of 1388)
937	Mixed-byte EBCDIC	Traditional Chinese
939	Mixed-byte EBCDIC	Japanese (subset of 5035)
1025	Single-byte EBCDIC	Cyrillic
1026	Single-byte EBCDIC	Turkish
1027	Single-byte EBCDIC	Japanese Latin (single-byte only)
1046	ASCII/ISO/Windows (bidirectional)	Windows Arabic ST5
1089	ASCII/ISO/Windows (bidirectional)	ISO 8859-6 (Arabic) ST5
1112	Single-byte EBCDIC	Baltic Multilingual
1122	Single-byte EBCDIC	Estonian
1123	Single-byte EBCDIC	Ukraine
1125	ASCII/ISO/Windows	Ukraine
1129	ASCII/ISO/Windows	Vietnamese
1130	Single-byte EBCDIC	Vietnamese
1131	ASCII/ISO/Windows	Belarus
1132	Single-byte EBCDIC	Lao
1140	Single-byte EBCDIC	United States and others (Euro support)
1141	Single-byte EBCDIC	Austria, Germany (Euro support)
1142	Single-byte EBCDIC	Denmark, Norway (Euro support)
1143	Single-byte EBCDIC	Finland, Sweden (Euro support)
1144	Single-byte EBCDIC	Italy (Euro support)
1145	Single-byte EBCDIC	Spain, Latin America (Euro support)
1146	Single-byte EBCDIC	United Kingdom (Euro support)
1147	Single-byte EBCDIC	France (Euro support)
1148	Single-byte EBCDIC	Latin-1 (MNCS) (Euro support)
1149	Single-byte EBCDIC	Iceland (Euro support)
1200	Unicode	Unicode UCS-2 (little-endian)
1250	ASCII/ISO/Windows	Windows Latin-2
1251	ASCII/ISO/Windows	Windows Cyrillic
1252	ASCII/ISO/Windows	Windows Latin-1
1253	ASCII/ISO/Windows	Windows Greek
1254	ASCII/ISO/Windows	Windows Turkey
1255	ASCII/ISO/Windows (bidirectional)	Windows Hebrew ST5
1256	ASCII/ISO/Windows (bidirectional)	Windows Arabic ST5
1257	ASCII/ISO/Windows	Windows Baltic
1258	ASCII/ISO/Windows	Windows Vietnam
1364	Mixed-byte EBCDIC	Japanese
1388	Mixed-byte EBCDIC	Simplified Chinese
1399	Mixed-byte EBCDIC	Japanese (in V4R5 and higher)

CCSID	Format	Description
4396	Double-byte EBCDIC	Japanese (subset of 300)
4930	Double-byte EBCDIC	Korean
4931	Double-byte EBCDIC	Traditional Chinese (subset of 835)
4933	Double-byte EBCDIC	Simplified Chinese GBK Graphic
4948	ASCII/ISO/Windows	Latin-2 (subset of 852)
4951	ASCII/ISO/Windows	Cyrillic (subset of 855)
5026	Mixed-byte EBCDIC	Japanese
5035	Mixed-byte EBCDIC	Japanese
5123	Single-byte EBCDIC	Japanese (single-byte only, Euro support)
5351	ASCII/ISO/Windows (bidirectional)	Windows Hebrew (Euro support) ST5
8492	Double-byte EBCDIC	Japanese (subset of 300)
8612	Single-byte EBCDIC	Arabic EBCDIC ST5
9026	Double-byte EBCDIC	Korean (subset of 834)
9029	Double-byte EBCDIC	Simplified Chinese (subset of 4933)
9066	ASCII/ISO/Windows	Thai (SBCS extended)
12588	Double-byte EBCDIC	Japanese (subset of 300)
13122	Double-byte EBCDIC	Korean (subset of 834)
16684	Double-byte EBCDIC	Japanese (available in V4R5)
17218	Double-byte EBCDIC	Korean (subset of 834)
12708	Single-byte EBCDIC	Arabic EBCDIC ST7
13488	Unicode	Unicode UCS-2 (big-endian)
28709	Single-byte EBCDIC	Traditional Chinese (single-byte only)
61952	Unicode	iSeries Unicode (used primarily in the integrated file system)
62211	Single-byte EBCDIC	Hebrew EBCDIC ST5
62224	Single-byte EBCDIC	Arabic EBCDIC ST6
62235	Single-byte EBCDIC	Hebrew EBCDIC ST6
62245	Single-byte EBCDIC	Hebrew EBCDIC ST10

CommandHelpRetriever class

The CommandHelpRetriever class retrieves help text for i5/OS control language (CL) commands and generates that text either in HTML or User Interface Manager (UIM) format. You can run CommandHelpRetriever from a command line or embed the functionality into your Java program.

To use CommandHelpRetriever, your server must run i5/OS V5R1 or later and have an XML parser and XSL processor in the CLASSPATH environment variable. For more information, see the following page:

"XML parser and XSLT processor" on page 393

Additionally, the Generate Command Documentation (GENCMDDOC) CL command uses the CommandHelpRetriever class. So you can simply use the GENCMDDOC command to take advantage of the functionality offered by the CommandHelpRetriever class. For more information, see the following page:

Generate Command Documentation (GENCMDDOC)

Running CommandHelpRetriever from a command line

You can run the CommandHelpRetriever class as a stand-alone command line program. To run CommandHelpRetriever from a command line, you must pass the following minimum parameters:

- The library on your iSeries server that contains the CL command. System commands reside in the QSYS library.
- The CL command.

You can also pass optional parameters to CommandHelpRetriever that include the iSeries server, the user ID, password, and the location for the generated file.

For more information, see the Javadoc reference documentation for CommandHelpRetriever.

Example: Using CommandHelpRetriever from a command line

The following example generates an HTML file called CRTLIB.html in the current directory.

Note: The example command appears on two lines for display purposes only. Type your command on a single line.

```
java com.ibm.as400.util.CommandHelpRetriever -library QSYS -command CRTLIB
     -system MySystem -userid MyUserID -password MyPassword
```

Embedding the CommandHelpRetriever class in your program

You can also use the CommandHelpRetriever class in your Java application to display the help documentation for specified CL commands. After you create a CommandHelpRetriever object, you can use the generateHTML and generateUIM methods to generate help documentation in either format.

When you use generateHTML(), you can display the generated HTML in the panel group for the command or you can specify a different panel group.

The following example creates a CommandHelpRetriever object and generates String objects that represent the HTML and UIM help documentation for the CRTLIB command.

```
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);
```

Javadoc reference documentation

For more information about the CommandHelpRetriever class, see the following Javadoc reference documentation:

CommandHelpRetriever

CommandPrompter class

The CommandPrompter class prompts for the parameter on a given command. The CommandPrompter offers functionality that is similar to the iSeries CL command prompt (pressing F4) and the same as the Management Central command prompt.

To use the CommandPrompter, the server must be running i5/OS V4R4 or later. For more information, see iSeries Navigator Information APARs and view the Required Fixes for Graphical Command Prompter Support.

Using CommandPrompter also requires that you have the following jar files in your CLASSPATH:

- jt400.jar
- jui400.jar
- util400.jar
- jhall.jar

You must also have an XML parser in your CLASSPATH. For more information about using a suitable XML parser, see the following page:

"XML parser and XSLT processor" on page 393

All of the jar files, except for jhall.jar, are included in IBM Toolbox for Java. For more information about IBM Toolbox for Java jar files, see Jar files. For more information about downloading jhall.jar, see the Sun JavaHelp(TM) Web site  .

To construct a CommandPrompter object, you pass it parameters for the parent frame that launches the prompter, the AS400 object on which the command will be prompted, and the command string. The command string can be a command name, a full command string, or a partial command name, such as `crt*`.

The CommandPrompter display is a modal dialog that the user must close before returning to the parent frame. The CommandPrompter handles any errors encountered during prompting. For a programming example that shows one way to use the CommandPrompter, see the following page:

"Example: Using CommandPrompter" on page 684

RunJavaApplication

The RunJavaApplication and VRunJavaApplication classes are utilities to run Java programs on the iSeries JVM. Unlike JavaApplicationCall and VJavaApplicationCall classes that you call from your Java program, RunJavaApplication and VRunJavaApplication are complete programs.

The RunJavaApplication class is a command line utility. It lets you set the environment (CLASSPATH and properties, for example) for the Java program. You specify the name of the Java program and its parameters, then you start the program. Once started, you can send input to the Java program which it receives via standard input. The Java program writes output to standard output and standard error.

The VRunJavaApplication utility has the same capabilities. The difference is VJavaApplicationCall uses a graphical user interface while JavaApplicationCall is a command line interface.

JPing

The JPing class is a command line utility that allows you to query your servers to see which services are running and which ports are in service. To query your servers from within a Java application, use the AS400JPing class.

See the JPing javadoc for more information about using JPing from within your Java application.

Call JPing from the command line by using the following syntax:

```
java utilities.JPing System [options]
```

where:

- System = the iSeries server that you want to query
- [options] = one or more of the available options

Options

You can use one or more of the following options. For options that have abbreviations, the abbreviation is listed in parenthesis.

-help (-h or -?)
Displays the help text.

-service *i5/OS_Service* **(-s** *i5/OS_Service***)**
Specifies one specific service to ping. The default action is to ping all services. You can use this option to specify one of the following services: as-file, as-netprt, as-rmtcmd, as-dtaq, as-database, as-ddm, as-central, and as-signon.

-ssl Specifies whether or not to ping the ssl ports. The default action is not to ping the ssl ports.

-timeout (-t)
Specifies the timeout period in milliseconds. The default setting is 20000, or 20 seconds.

Example: Using JPing from the command line

For example, use the following command to ping the as-dtaq service, including ssl ports, with a timeout of 5 seconds:

```
java utilities.JPing myServer -s as-dtaq -ssl -t 5000
```

Vaccess classes

The Vaccess package and its classes have been deprecated. You are advised to use the Access package in combination with Java Swing instead.

IBM Toolbox for Java provides a set of graphical user interface (GUI) classes in the vaccess package. These classes use the access classes to retrieve data and to present the data to the user.

Java programs that use the IBM Toolbox for Java vaccess classes require the Swing package, which comes with the Java 2 Platform, Standard Edition (J2SE). For more information about Swing, see the Sun Java Foundation Classes  Web site.

For more information about the relationships between the IBM Toolbox for Java GUI classes, the Access classes, and Java Swing, see the Vaccess classes diagram.

Use the AS400 panes classes to display iSeries data.

APIs are available to access the following iSeries resources and their tools:

- Command call
- Data queues
- Error events*
- Integrated file system
- JavaApplicationCall
- JDBC
- Jobs*
- Messages*
- Permission

- Print* including the spooled file viewer
- ProgramCall and ProgramParameter
- Record-level access
- Resource lists
- System status
- System values
- Users and Groups

Note: AS400 panes are used with other vaccess classes (see items marked above with an asterisk) to present and allow manipulation of iSeries resources.

When programming with the IBM Toolbox for Java graphical user interface components, use the Error events classes to report and handle error events to the user.

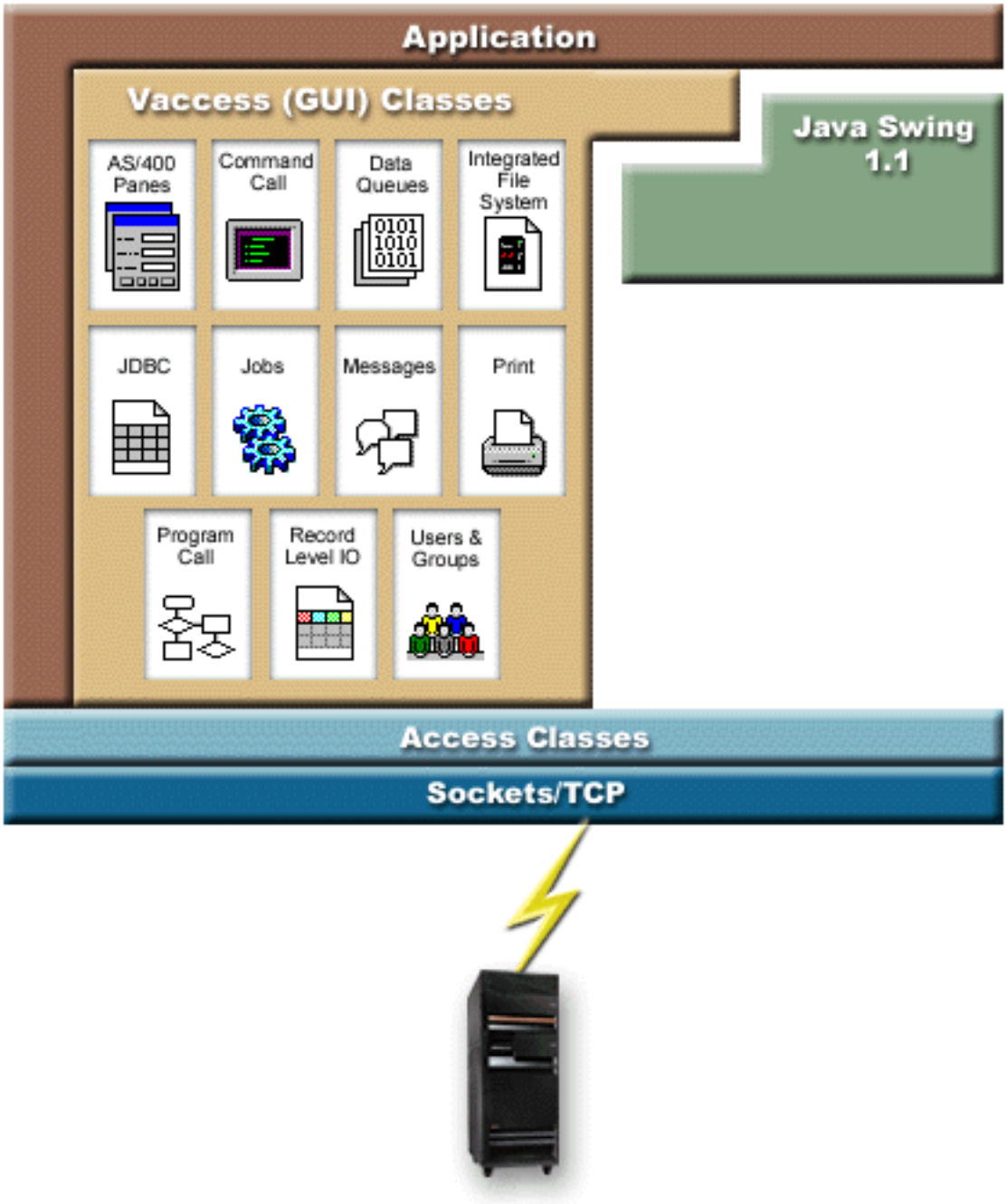
See Access classes for more information about accessing iSeries data.

Vaccess classes

The Vaccess package and its classes have been deprecated. You are advised to use the Access package in combination with Java Swing instead.

IBM Toolbox for Java provides graphical user interface (GUI) classes in the vaccess package to retrieve, display, and in some cases manipulate, server data. These classes use the Java Swing 1.1 framework. Figure 1 shows the relationship between these classes:

Figure 1: Vaccess classes



"Long description of Figure 1: Vaccess classes (rzahh508.gif)"

Long description of Figure 1: Vaccess classes (rzahh508.gif):

found in IBM Toolbox for Java: Vaccess package diagram

This figure shows the relationship between the classes in the vaccess package, the access package, and the Java Swing classes.

Description

The figure is composed of the following:

- The topmost image is actually a thick brown border, as of the left and top sides of a rectangle, labeled 'Application,' that represents a Java application. The rectangle delineated by the Java application contains the following irregularly shaped images that fit together like pieces of a puzzle:
- A tan polygon that represents the IBM Toolbox for Java vaccess (GUI) classes. This shape contains smaller images that represent the functions contained in the vaccess classes.
- A green polygon that represents the Java Swing classes
- Below these images is a light blue bar labeled Access Classes, that represents classes in the IBM Toolbox for Java access package.
- Below the light blue bar is an identically shaped dark blue bar, labeled 'Sockets/TCP.'
- The bottom image is a picture of an iSeries server.
- A lightning bolt, which represents a socket connection from the Java application to the server, extends downward from Sockets/TCP (the dark blue bar) and connects to the server (the image of an iSeries server).

A Java application (the area delineated by the thick brown border) contains vaccess classes (the tan polygon) and Java Swing classes (the green polygon). The vaccess classes enable the Java application to access the following data and functions on the server (small images in the tan polygon):

AS400Panes, CommandCall, DataQueues, integrated file system, JDBC, jobs, messages, print, ProgramCall, record-level input and output, and users and groups

The Java application uses IBM Toolbox for Java access classes (the light blue bar) to create one or more socket connections (the dark blue bar). The socket connections enable the Java application to communicate (the lightning bolt) with the server (bottom image of the iSeries server).

AS400Panes

AS400Panes are components in the vaccess package that present and allow manipulation of one or more server resources in a GUI. The behavior of each server resource varies depending on the type of resource.

All panes extend the Java Component class. As a result, they can be added to any AWT Frame, Window, or Container.

The following AS400Panes are available:

- AS400DetailsPane presents a list of server resources in a table where each row displays various details about a single resource. The table allows selection of one or more resources.
- AS400ExplorerPane combines an AS400TreePane and AS400DetailsPane so that the resource selected in the tree is presented in the details.
- AS400JDBCDataSourcePane presents the property values of an AS400JDBCDataSource object.
- AS400ListPane presents a list of server resources and allows selection of one or more resources.
- AS400TreePane presents a tree hierarchy of server resources and allows selection of one or more resources.

Server resources

Server resources are represented in the graphical user interface with an icon and text. Server resources are defined with hierarchical relationships where a resource might have a parent and zero or more children. These are predefined relationships and are used to specify what resources are displayed in an AS400Pane. For example, VJobList is the parent to zero or more VJobs, and this hierarchical relationship is represented graphically in an AS400Pane.

The IBM Toolbox for Java provides access to the following server resources:

- VIFSDirectory represents a directory in the integrated file system

- VJob and VJobList represent a job or a list of jobs
- VMessageList and VMessageQueue represent a list of messages returned from a CommandCall or ProgramCall or a message queue
- VPrinter, VPrinters, and VPrinterOutput represent a printer, a list of printers, or a list of spooled files
- VUserList represents a list of users

All resources are implementations of the VNode interface.

Setting the root

To specify which server resources are presented in an AS400Pane, set the root using the constructor or setRoot() method. The root defines the top level object and is used differently based on the pane:

- AS400ListPane presents all of the root's children in its list
- AS400DetailsPane presents all of the root's children in its table
- AS400TreePane uses the root as the root of its tree
- AS400ExplorerPane uses the root as the root of its tree

Any combination of panes and roots is possible.

The following example creates an AS400DetailsPane to present the list of users defined on the system:

```
                    // 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);
```

Loading the contents

When AS400Pane objects and server resource objects are created, they are initialized to a default state. The relevant information that makes up the contents of the pane is not loaded at creation time.

To load the contents, the application must explicitly call the load() method. In most cases, this initiates communication to the server to gather the relevant information. Because it can sometimes take a while to gather this information, the application can control exactly when it happens. For example, you can:

- Load the contents before adding the pane to a frame. The frame does not appear until all information is loaded.
- Load the contents after adding the pane to a frame and displaying that frame. The frame appears, but it does not contain much information. A ″wait cursor″ appears and the information is filled in as it is loaded.

The following example loads the contents of a details pane before adding it to a frame:

```
                    // 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);
```

Actions and properties panes

At run time, the user can select a pop-up menu on any server resource. The pop-up menu presents a list of relevant actions that are available for the resource. When the user selects an action from the pop-up menu, that action is performed. Each resource has different actions defined.

In some cases, the pop-up menu also presents an item that allows the user to view a properties pane. A properties pane shows various details about the resource and may allow the user to change those details.

The application can control whether actions and properties panes are available by using the setAllowActions() method on the pane.

Models

The AS400Panes are implemented using the model-view-controller paradigm, in which the data and the user interface are separated into different classes. The AS400Panes integrate IBM Toolbox for Java models with Java GUI components. The models manage server resources and the vaccess components display them graphically and handle user interaction.

The AS400Panes provide enough functionality for most requirements. However, if an application needs more control of the JFC component, then the application can access a server model directly and provide customized integration with a different vaccess component.

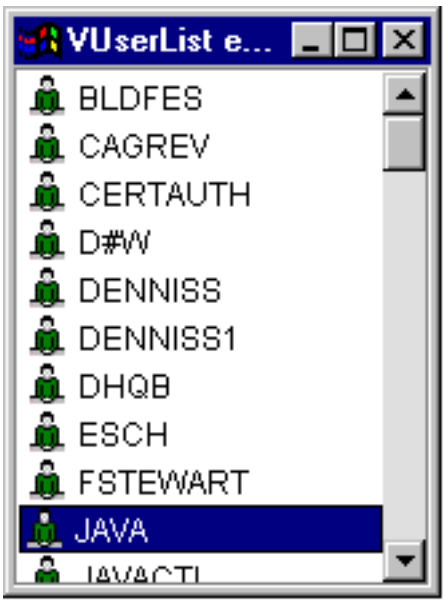
The following models are available:

- AS400ListModel implements the JFC ListModel interface as a list of server resources. This can be used with a JFC JList object.
- AS400DetailsModel implements the JFC TableModel interface as a table of server resources where each row contains various details about a single resource. This can be used with a JFC JTable object.
- AS400TreeModel implements the JFC TreeModel interface as a tree hierarchy of server resources. This can be used with a JFC JTree object.

Examples

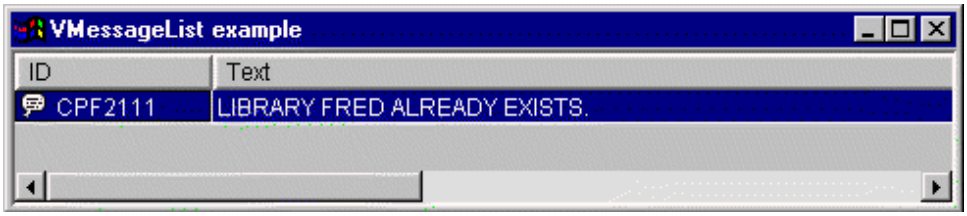
- Present a list of users on the system using an AS400ListPane with a VUserList object. Figure 1 shows the finished product:

  Figure 1: Using AS400ListPane with a VUserList object



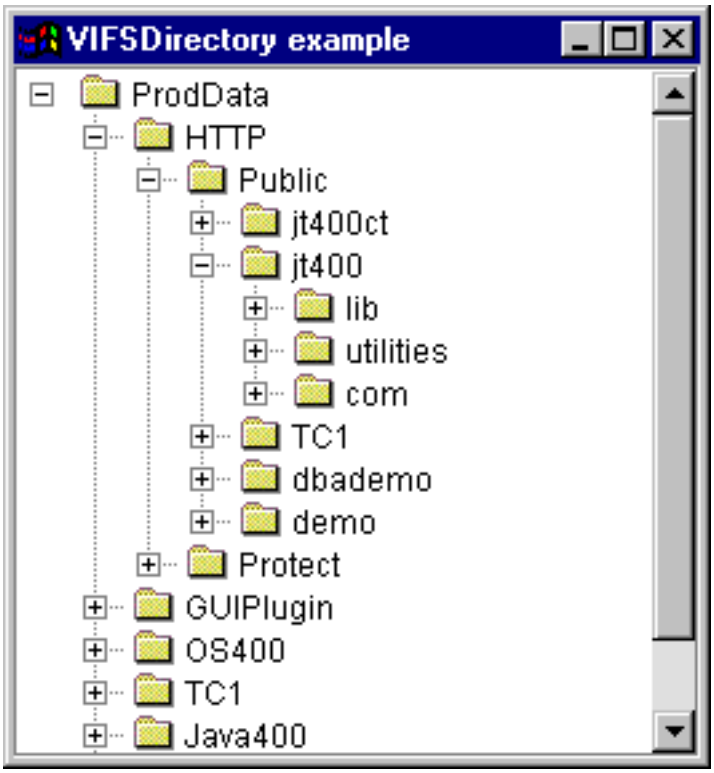
- Present the list of messages generated by a command call using an AS400DetailsPane with a VMessageList object. Figure 2 shows the finished product:

  Figure 2: Using AS400DetailsPane with a VMessageList object



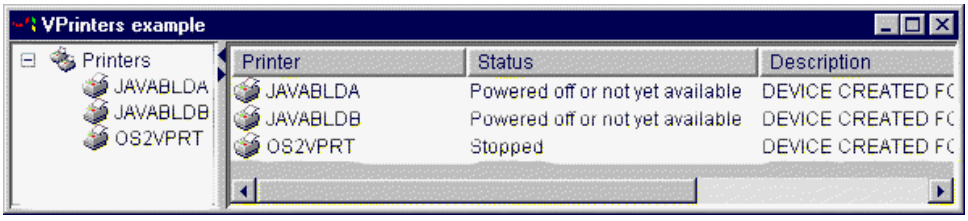
- Present an integrated file system directory hierarchy using an AS400TreePane with a VIFSDirectory object. Figure 3 shows the finished product:

  Figure 3: Using AS400TreePane with a VIFSDirectory object



- Present print resources using an AS400ExplorerPane with a VPrinters object. Figure 4 shows the finished product:

  Figure 4: Using AS400ExplorerPane with a VPrinters object



Command Call

The command call vaccess (GUI) components allow a Java program to present a button or menu item that calls a non-interactive server command.

A CommandCallButton object represents a button that calls a server command when pressed. The CommandCallButton class extends the Java Foundation Classes (JFC) JButton class so that all buttons have a consistent appearance and behavior.

Similarly, a CommandCallMenuItem object represents a menu item that calls a server command when selected. The CommandCallMenuItem class extends the JFC JMenuItem class so that all menu items also have a consistent appearance and behavior.

To use a command call graphical user interface component, set both the system and command properties. These properties can be set using a constructor or through the setSystem() and setCommand() methods.

The following example creates a CommandCallButton. At run time, when the button is pressed, it creates a library called "FRED":

```
// 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);
```

When a server command runs, it may return zero or more server messages. To detect when the server command runs, add an ActionCompletedListener to the button or menu item using the addActionCompletedListener() method. When the command runs, it fires an ActionCompletedEvent to all such listeners. A listener can use the getMessageList() method to retrieve any server messages that the command generated.

This example adds an ActionCompletedListener that processes all server messages that the command generated:

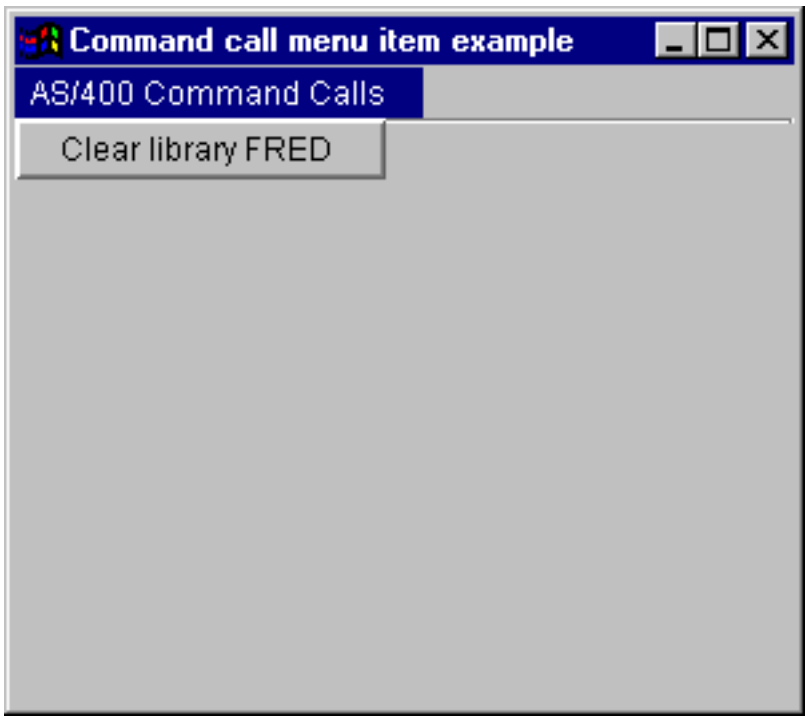
```
                    // Add an ActionCompletedListener that
                    // is implemented using an anonymous
                    // inner class. This is a convenient
                    // way to specify simple event
                    // listeners.
button.addActionCompletedListener (new 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.
     }
});
```

Examples

This example shows how to use a CommandCallMenuItem in an application.

Figure 1 shows the CommandCall graphical user interface component:

Figure 1: CommandCall GUI component



Data queues

The data queue graphical components allow a Java program to use any Java Foundation Classes (JFC) graphical text component to read or write to a server data queue.

The DataQueueDocument and KeyedDataQueueDocument classes are implementations of the JFC Document interface. These classes can be used directly with any JFC graphical text component. Several text components, such as single line fields (JTextField) and multiple line text areas (JTextArea), are available in JFC.

Data queue documents associate the contents of a text component with a server data queue. (A text component is a graphical component used to display text that the user can optionally edit.) The Java program can read and write between the text component and data queue at any time. Use DataQueueDocument for **sequential** data queues and KeyedDataQueueDocument for **keyed** data queues.

To use a DataQueueDocument, set both the system and path properties. These properties can be set using a constructor or through the setSystem() and setPath() methods. The DataQueueDocument object is then "plugged" into the text component, usually using the text component's constructor or setDocument() method. KeyedDataQueueDocuments work the same way.

The following example creates a DataQueueDocument whose contents are associated with a data queue:

```
                    // 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);
```

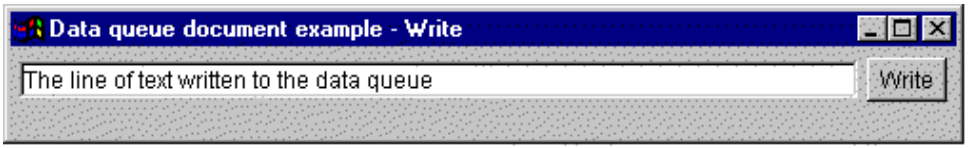
Initially, the contents of the text component are empty. Use read() or peek() to fill the contents with the next entry on the queue. Use write() to write the contents of the text component to the data queue. Note that these documents only work with String data queue entries.

Examples

Example of using a DataQueueDocument in an application.

Figure 1 shows the DataQueueDocument graphical user interface component being used in a JTextField. A button has been added to provide a GUI interface for the user to write the contents of the test field to the data queue.

Figure 1: DataQueueDocument GUI component



Error events

In most cases, the IBM Toolbox for Java GUI components fire error events instead of throw exceptions.

An error event is a wrapper around an exception that is thrown by an internal component.

You can provide an error listener that handles all error events that are fired by a particular graphical user interface component. Whenever an exception is thrown, the listener is called, and it can provide appropriate error reporting. By default, no action takes place when error events are fired.

The IBM Toolbox for Java provides a graphical user interface component called ErrorDialogAdapter, which automatically displays a dialog to the user whenever an error event is fired.

Examples

The following examples show how you can handle errors and define a simple error listener.

Example: Handling error events by displaying a dialog

The following example shows how you can handle error events by displaying a dialog:

```
// 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);
```

Example: Defining an error listener

You can write a custom error listener to handle errors in a different way. Use the ErrorListener interface to accomplish this.

The following example shows how to define an simple error listener that only prints errors to System.out:

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

Example: Handling error events by using an error listener

The following example shows how to handle error events for a graphical user interface component using this customized handler:

```
MyErrorHandler errorHandler = new MyErrorHandler ();
component.addErrorListener (errorHandler);
```

Related reference

"Vaccess classes" on page 240
The Vaccess package and its classes have been deprecated. You are advised to use the Access package in combination with Java Swing instead.

"Exceptions" on page 48
The IBM Toolbox for Java access classes throw exceptions when device errors, physical limitations, programming errors, or user input errors occur. The exception classes are based upon the type of error that occurs instead of the location where the error originates.

Integrated file system

The integrated file system graphical user interface components allow a Java program to present directories and files in the integrated file system on the server in a GUI.

The following components are available:

- IFSFileSystemView provides a gateway to the iSeries integrated file system.
- IFSFileDialog presents a dialog that allows the user to choose a directory and select a file by navigating through the directory hierarchy.
- VIFSDirectory is a resource that represents a directory in the integrated file system for use in AS400Panes.
- IFSTextFileDocument represents a text file for use in any Java Foundation Classes (JFC) graphical text component.
- To use the integrated file system graphical user interface components, set both the system and the path or directory properties. These properties can be set using a constructor or through the setDirectory() (for IFSFileDialog) or setSystem() and setPath() methods (for VIFSDirectory and IFSTextFileDocument).

Set the path to something other than "/QSYS.LIB" because this directory is typically large, and downloading its contents can take a long time.

IFSFileSystemView:

This class has been deprecated and replaced by class com.ibm.as400.access.IFSSystemView.

The IFSFileSystemView provides a gateway to the iSeries integrated file system, for use when constructing javax.swing.JFileChooser objects.

JFileChooser is a standard Java way to build dialogs for navigating and choosing files, and is the recommended replacement for IFSFileDialog.

Example: Using IFSFileSystemView

The following example demonstrates the use of IFSFileSystemView.

```
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());
}
```

File dialogs:

The IFSFileDialog class is a dialog that allows the user to traverse the directories of the integrated file system on the server and select a file. The caller can set the text on the buttons on the dialog. In addition, the caller can use FileFilter objects, which allow the user to limit the choices to certain files.

If the user selects a file in the dialog, use the getFileName() method to get the name of the selected file. Use the getAbsolutePath() method to get the full path name of the selected file.

The following example sets up an integrated file system file dialog with two file filters:

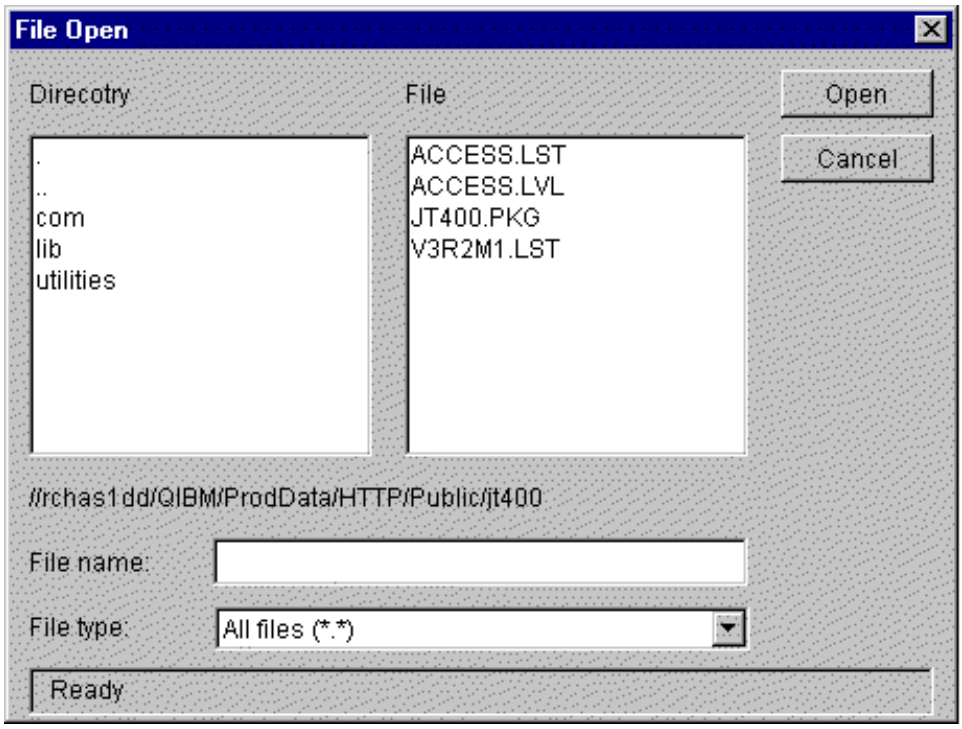
```
                  // Create a IFSFileDialog object
                  // setting the text of the title bar.
                  // Assume that "system" is an AS400
                  // object and "frame" is a JFrame
                  // created and initialized elsewhere.
IFSFileDialog dialog = new IFSFileDialog (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 ());
```

Example

Present an IFSFileDialog and print the selection, if any.

Figure 1 shows the IFSFileDialog graphical user interface component:

Figure 1: IFSFileDialog GUI component



Directories in AS400Panes:

AS400Panes are GUI components that present and allow manipulation of one or more server resources. A VIFSDirectory object is a resource that represents a directory in the integrated file system for use in AS400Panes. AS400Pane and VIFSDirectory objects can be used together to present many views of the integrated file system, and to allow the user to navigate, manipulate, and select directories and files.

To use a VIFSDirectory, set both the system and path properties. You set these properties using a constructor or through the setSystem() and setPath() methods. You then plug the VIFSDirectory object into the AS400Pane as the root, using the constructor or setRoot() method of the AS400Pane.

VIFSDirectory has some other useful properties for defining the set of directories and files that are presented in AS400Panes. Use setInclude() to specify whether directories, files, or both appear. Use setPattern() to set a filter on the items that are shown by specifying a pattern that the file name must match. You can use wildcard characters, such as "*" and "?", in the patterns. Similarly, use setFilter() to set a filter with an IFSFileFilter object.

When AS400Pane objects and VIFSDirectory objects are created, they are initialized to a default state. The subdirectories and the files that make up the contents of the root directory have not been loaded. To load the contents, the caller must explicitly call the load() method on either object to initiate communication to the server to gather the contents of the directory.

At run-time, a user can perform actions on any directory or file by right-clicking it to display the context menu. The directory context menu can include the following items:

- **Create file** - creates a file in the directory. This will give the file a default name
- **Create directory** - creates a subdirectory with a default name
- **Rename** - renames a directory
- **Delete** - deletes a directory
- **Properties** - displays properties such as the location, number of files and subdirectories, and modification date

The file context menu can include the following items:

- **Edit** - edits a text file in a different window
- **View** - views a text file in a different window
- **Rename** - renames a file
- **Delete** - deletes a file
- **Properties** - displays properties such as the location, size, modification date, and attributes

Users can only read or write directories and files to which they are authorized. In addition, the caller can prevent the user from performing actions by using the setAllowActions() method on the pane.

The following example creates a VIFSDirectory and presents it in an AS400ExplorerPane:

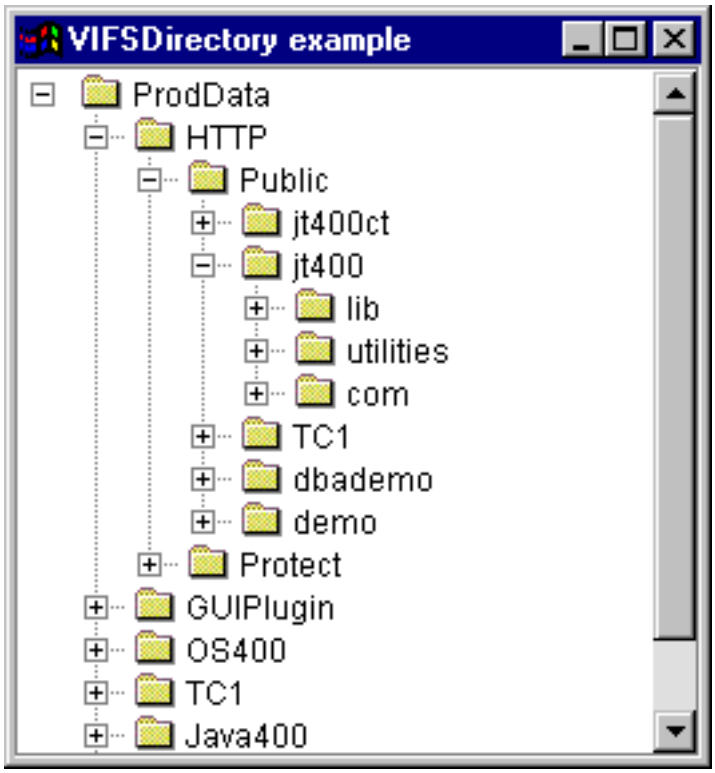
```
                    // Create the VIFSDirectory object.
                    // Assume that "system" in an AS400
                    // object created and initialized
                    // elsewhere.
    VIFSDirectory root = new VIFSDirectory (system, "/DirectoryA/DirectoryB");

                    // 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);
```

Example

Present an integrated file system directory hierarchy using an AS400TreePane with a VIFSDirectory object.

Figure 1 shows the VIFSDirectory graphical user interface component:

Figure 1: VIFSDirectory GUI component



IFSTextFileDocument:

Text file documents allow a Java program to use any Java Foundation Classes (JFC) graphical text component to edit or view text files in the integrated file system on a server. (A text component is a graphical component used to display text that the user can optionally edit.)

The IFSTextFileDocument class is an implementation of the JFC Document interface. It can be used directly with any JFC graphical text component. Several text components, such as single line fields (JTextField) and multiple line text areas (JTextArea), are available in JFC.

Text file documents associate the contents of a text component with a text file. The Java program can load and save between the text component and the text file at any time.

To use an IFSTextFileDocument, set both the system and path properties. These properties can be set using a constructor or through the setSystem() and setPath() methods. The IFSTextFileDocument object is then ″plugged″ into the text component, typically using the text component's constructor or setDocument() method.

Initially, the contents of the text component are empty. Use load() to load the contents from the text file. Use save() to save the contents of the text component to the text file.

The following example creates and loads an IFSTextFileDocument:

```
                    // 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);
```

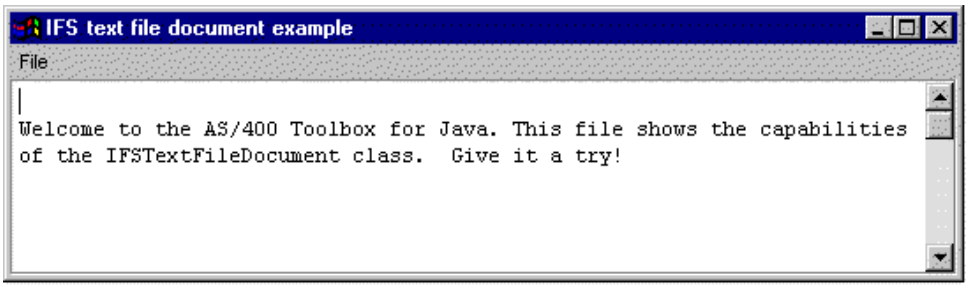
```
                    // Add the text area to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
    frame.getContentPane ().add (textArea);
```

Example

Present an IFSTextFileDocument in a JTextPane.

Figure 1 shows the IFSTextFileDocument graphical user interface component:

Figure 1: IFSTextFileDocument example



VJavaApplicationCall class

The VJavaApplicationCall class allows you to run a Java application on the server from a client by using a graphical user interface (GUI).

The GUI is a panel with two sections. The top section is an output window that displays output that the Java program writes to standard output and standard error. The bottom section is an input field where the user enters the Java environment, the Java program to run with parameters and input the Java program receives via standard input. Refer to the Java command options for more information.

For example, this code might create the following GUI for your Java program.

VJavaApplicationCall is a class that you call from your Java program. However, the IBM Toolbox for Java also provides a utility that is a complete Java application that can be used to call your Java program from a workstation. Refer to the RunJavaApplication class for more information.

JDBC classes

The JDBC graphical user interface components allow a Java program to present various views and controls for accessing a database using SQL (Structured Query Language) statements and queries.

The following components are available:

- SQLStatementButton and SQLStatementMenuItem are either a button or a menu item that issues an SQL statement when clicked or selected.
- SQLStatementDocument is a document that can be used with any Java Foundation Classes (JFC) graphical text component to issue an SQL statement.
- SQLResultSetFormPane presents the results of an SQL query in a form.
- SQLResultSetTablePane presents the results of an SQL query in a table.
- SQLResultSetTableModel manages the results of an SQL query in a table.
- SQLQueryBuilderPane presents an interactive tool for dynamically building SQL queries.

All JDBC graphical user interface components communicate with the database using a JDBC driver. The JDBC driver must be registered with the JDBC driver manager in order for any of these components to work. The following example registers the IBM Toolbox for Java JDBC driver:

```
                    // Register the JDBC driver.
DriverManager.registerDriver (new com.ibm.as400.access.AS400JDBCDriver ());
```

SQL connections

An SQLConnection object represents a connection to a database using JDBC. **The SQLConnection object is used with all of the JDBC graphical user interface components.**

To use an SQLConnection, set the URL property using the constructor or setURL(). This identifies the database to which the connection is made. Other optional properties can be set:

- Use setProperties() to specify a set of JDBC connection properties.
- Use setUserName() to specify the user name for the connection.
- Use setPassword() to specify the password for the connection.

The actual connection to the database is not made when the SQLConnection object is created. Instead, it is made when getConnection() is called. This method is normally called automatically by the JDBC graphical user interface components, but it can be called at any time in order to control when the connection is made.

The following example creates and initializes an SQLConnection object:

```
                    // 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");
```

An SQLConnection object can be used for more than one JDBC graphical user interface component. All such components will use the same connection, which can improve performance and resource usage. Alternately, each JDBC graphical user interface component can use a different SQL object. It is sometimes necessary to use separate connections, so that SQL statements are issued in different transactions.

When the connection is no longer needed, close the SQLConnection object using close(). This frees up JDBC resources on both the client and server.

Buttons and menu items:

An SQLStatementButton object represents a button that issues an SQL (Structured Query Language) statement when pressed. The SQLStatementButton class extends the Java Foundation Classes (JFC) JButton class so that all buttons have a consistent appearance and behavior.

Similarly, an SQLStatementMenuItem object represents a menu item that issues an SQL statement when selected. The SQLStatementMenuItem class extends the JFC JMenuItem class so that all menu items have a consistent appearance and behavior.

To use either of these classes, set both the connection and the SQLStatement properties. These properties can be set using a constructor or the setConnection() and setSQLStatement() methods.

The following example creates an SQLStatementButton. When the button is pressed at run time, it deletes all records in a table:

```
                    // 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);
```

After the SQL statement is issued, use getResultSet(), getMoreResults(), getUpdateCount(), or getWarnings() to retrieve the results.

SQLStatementDocument class:

The SQLStatementDocument class is an implementation of the Java Foundation Classes (JFC) Document interface. It can be used directly with any JFC graphical text component. Several text components, such as single line fields (JTextField) and multiple line text areas (JTextArea), are available in JFC. SQLStatementDocument objects associate the contents of text components with SQLConnection objects. The Java program can run the SQL statement contained in the document contents at any time and then process the results, if any.

To use an SQLStatementDocument, you must set the connection property. Set this property by using the constructor or the setConnection() method. The SQLStatementDocument object is then ″plugged″ into the text component, typically using the text component's constructor or setDocument() method. Use execute() at any time to run the SQL statement contained in the document.

The following example creates an SQLStatementDocument in a JTextField:

```
                  // 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 ();
```

After the SQL statement is issued, use getResultSet(), getMoreResults(), getUpdateCount(), or getWarnings() to retrieve the results.

SQLResultSetFormPane class:

An SQLResultSetFormPane presents the results of an SQL (Structured Query Language) query in a form. The form displays one record at a time and provides buttons that allow the user to scroll forward, backward, to the first or last record, or refresh the view of the results.

To use an SQLResultSetFormPane, set the connection and query properties. Set these properties by using the constructor or the setConnection() and setQuery() methods. Use load() to execute the query and present the first record in the result set. When the results are no longer needed, call close() to ensure that the result set is closed.

The following example creates an SQLResultSetFormPane object and adds it to a frame:

```
                    // Create an SQLResultSetFormPane
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection, "
                                SELECT * FROM QIWS.QCUSTCDT");
                    // Load the results.
formPane.load ();

                    // Add the form pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (formPane);
```

SQLResultSetTablePane class:

An SQLResultSetTablePane presents the results of an SQL (Structured Query Language) query in a table. Each row in the table displays a record from the result set and each column displays a field.

To use an SQLResultSetTablePane, set the connection and query properties. Set properties by using the constructor or the setConnection() and setQuery() methods. Use load() to execute the query and present the results in the table. When the results are no longer needed, call close() to ensure that the result set is closed.

The following example creates an SQLResultSetTablePane object and adds it to a frame:

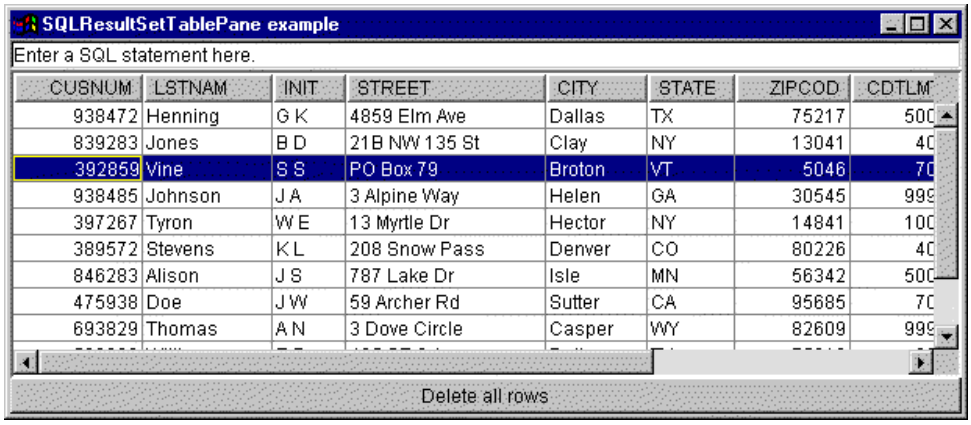
```
                    // Create an SQLResultSetTablePane
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
SQLResultSetTablePane tablePane = new SQLResultSetTablePane (connection,
                                  "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);
```

Example

Present an SQLResultSetTablePane that displays the contents of a table. This example uses an SQLStatementDocument (denoted in the following image by the text, ″Enter a SQL statement here″) that allows the user to type in any SQL statement, and an SQLStatementButton (denoted by the text, ″Delete all rows″) that allows the user to delete all rows from the table.

Figure 1 shows the SQLResultSetTablePane graphical user interface component:

Figure 1: SQLResultSetTablePane GUI component



SQLResultSetTableModel class:

SQLResultSetTablePane is implemented using the model-view-controller paradigm, in which the data and the user interface are separated into different classes. The implementation integrates SQLResultSetTableModel with the Java Foundation Classes' (JFC) JTable. The SQLResultSetTableModel class manages the results of the query and JTable displays the results graphically and handles user interaction.

SQLResultSetTablePane provides enough functionality for most requirements. However, if a caller needs more control of the JFC component, then the caller can use SQLResultSetTableModel directly and provide customized integration with a different graphical user interface component.

To use an SQLResultSetTableModel, set the connection and query properties. Set these properties by using the constructor or the setConnection() and setQuery() methods. Use load() to execute the query and load the results. When the results are no longer needed, call close() to ensure that the result set is closed.

The following example creates an SQLResultSetTableModel object and presents it with a JTable:

```
                    // Create an SQLResultSetTableModel
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
SQLResultSetTableModel tableModel = new SQLResultSetTableModel (connection,
                                    "SELECT * FROM QIWS.QCUSTCDT");
                    // Load the results.
tableModel.load ();

                    // Create a JTable for the model.
JTable table = new JTable (tableModel);

                    // Add the table to a frame.  Assume
                    // that "frame" is a JFrame created
                    // elsewhere.
frame.getContentPane ().add (table);
```

SQL query builders:

An SQLQueryBuilderPane presents an interactive tool for dynamically building SQL queries.

To use an SQLQueryPane, set the connection property. This property can be set using the constructor or the setConnection() method. Use load() to load data needed for the query builder graphical user interface. Use getQuery() to get the SQL query that the user has built.

The following example creates an SQLQueryBuilderPane object and adds it to a frame:

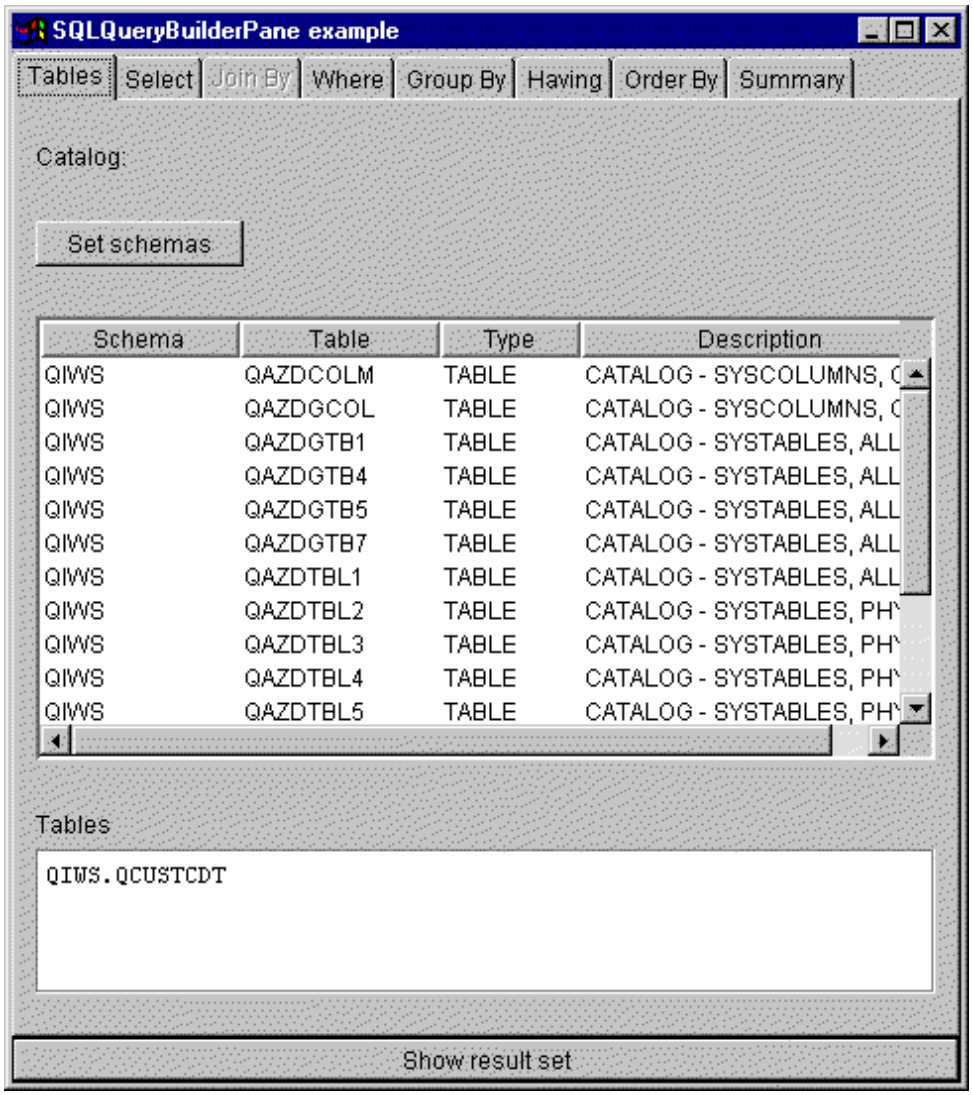
```
                    // Create an SQLQueryBuilderPane
                    // object. Assume that "connection"
                    // is an SQLConnection object that is
                    // created and initialized elsewhere.
SQLQueryBuilderPane queryBuilder = new SQLQueryBuilderPane (connection);

                    // 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);
```

Example

Present an SQLQueryBuilderPane and a button. When the button is clicked, present the results of the query in an SQLResultSetFormPane in another frame.

Figure 1 shows the SQLQueryBuilderPane graphical user interface component:

Figure 1: SQLQueryBuilderPane GUI component



Jobs

The jobs vaccess (GUI) components allow a Java program to present lists of server jobs and job log messages in a GUI.

The following components are available:

- A VJobList object is a resource that represents a list of server jobs for use in AS400Panes.
- A VJob object is a resource that represents the list of messages in a job log for use in AS400Panes.

You can use AS400Panes, VJobList objects, and VJob objects together to present many views of a job list or job log.

To use a VJobList, set the system, name, number, and user properties. Set these properties by using a constructor or through the setSystem(), setName(), setNumber(), and setUser() properties.

To use a VJob, set the system property. Set this property by using a constructor or through the setSystem() method.

Either the VJobList or VJob object is then ″plugged″ into the AS400Pane as the root, using the pane's constructor or setRoot() method.

VJobList has some other useful properties for defining the set of jobs that are presented in AS400Panes. Use setName() to specify that only jobs with a certain name appear. Use setNumber() to specify that only jobs with a certain number appear. Similarly, use setUser() to specify that only jobs for a certain user appear.

When AS400Pane, VJobList, and VJob objects are created, they are initialized to a default state. The list of jobs or job log messages are not loaded at creation time. To load the contents, the caller must explicitly call the load() method on either object. This will initiate communication to the server to gather the contents of the list.

At run-time, right-click a job, job list, or job log message to display the shortcut menu. Select **Properties** from the shortcut menu to perform actions on the selected object:

- Job - Work with properties, such as the type and status. You can also change the value of some of the properties.
- Job list - Work with the properties, such as name, number, and user properties. You can also change the contents of the list.
- Job log message - Display properties, such as the full text, severity, and time sent.

Users can only access jobs to which they are authorized. In addition, the Java program can prevent the user from performing actions by using the setAllowActions() method on the pane.

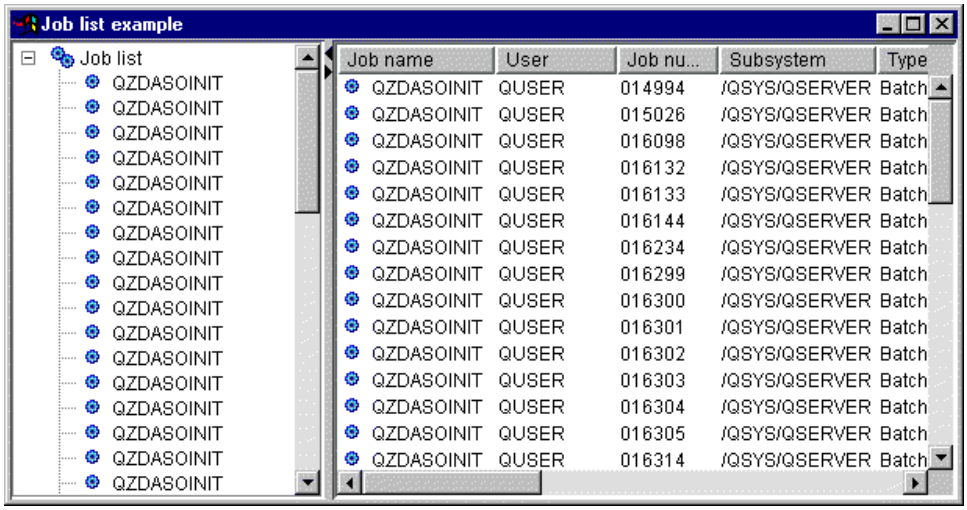
The following example creates a VJobList and presents it in an AS400ExplorerPane:

```
                    // 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);
```

Examples

This VJobList example presents an AS400ExplorerPane filled with a list of jobs. The list shows jobs on the system that have the same job name.

The following image shows the VJobList graphical user interface component:



Vaccess message classes

The messages graphical user interface components allow a Java program to present lists of server messages in a GUI.

The following components are available:

- A Message list object is a resource that represents a list of messages for use in AS400Panes. This is for message lists generated by command or program calls.
- A Message queues object is a resource that represents the messages in a server message queue for use in AS400Ppanes.

AS400Panes are graphical user interface components that present and allow manipulation of one or more server resources. VMessageList and VMessageQueue objects are resources that represent lists of server messages in AS400Panes.

You can use AS400Pane, VMessageList, and VMessageQueue objects together to present many views of a message list and to allow the user to select and perform operations on messages.

VMessageList class:

A VMessageList object is a resource that represents a list of messages for use in AS400Panes. This is for message lists generated by command or program calls. The following methods return message lists:

- CommandCall.getMessageList()
- CommandCallButton.getMessageList()
- CommandCallMenuItem.getMessageList()
- ProgramCall.getMessageList()
- ProgramCallButton.getMessageList()
- ProgramCallMenuItem.getMessageList()

To use a VMessageList, set the messageList property. Set this property by using a constructor or through the setMessageList() method. The VMessageList object is then ″plugged″ into the AS400Pane as the root, using the constructor or setRoot() method of the AS400Pane.

When AS400Pane and VMessageList objects are created, they are initialized to a default state. The list of messages is not loaded at creation time. To load the contents, the caller must explicitly call the load() method on either object.

At run-time, a user can perform actions on a message by right-clicking it to display the context menu. The message context menu can include an item called **Properties** that displays properties such as the severity, type, and date.

The caller can prevent the user from performing actions by using the setAllowActions() method on the pane.

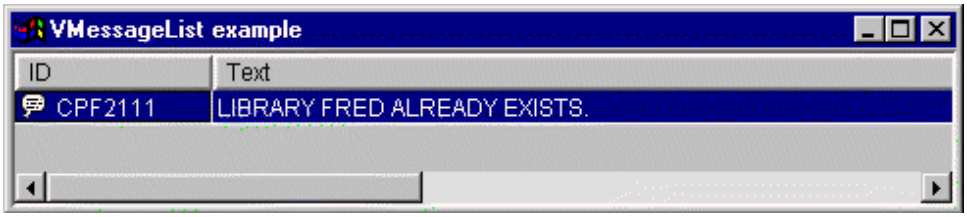
The following example creates a VMessageList for the messages generated by a command call and presents it in an AS400DetailsPane:

```
                    // 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);
```

Example

Present the list of messages generated by a command call using an AS400DetailsPane with a VMessageList object. Figure 1 shows the VMessageList graphical user interface component:

Figure 1: VMessageList GUI component



VMessageQueue class:

A VMessageQueue object is a resource that represents the messages in a server message queue for use in AS400Panes.

To use a VMessageQueue, set the system and path properties. These properties can be set using a constructor or through the setSystem() and setPath() methods. The VMessageQueue object is then ″plugged″ into the AS400Pane as the root by using the constructor or setRoot() method of the AS400Pane.

VMessageQueue has some other useful properties for defining the set of messages that are presented in AS400Panes. Use setSeverity() to specify the severity of messages that appear. Use setSelection() to specify the type of messages that appear.

When AS400Pane and VMessageQueue objects are created, they are initialized to a default state. The list of messages is not loaded at creation time. To load the contents, the caller must explicitly call the load() method on either object. This will initiate communication to the server to gather the contents of the list.

At run-time, a user can perform actions on a message or message queue by right-clicking it to display the context menu. The context menu for message queues can include the following items:

- **Clear** - clears the message queue
- **Properties** - allows the user to set the severity and selection properties. This may be used to change the contents of the list

The following action is available for messages on a message queue:

- **Remove** - removes the message from the message queue
- **Reply** - replies to an inquiry message
- **Properties** - displays properties such as the severity, type, and date

Of course, users can only access message queues to which they are authorized. In addition, the caller can prevent the user from performing actions by using the setAllowActions() method on the pane.

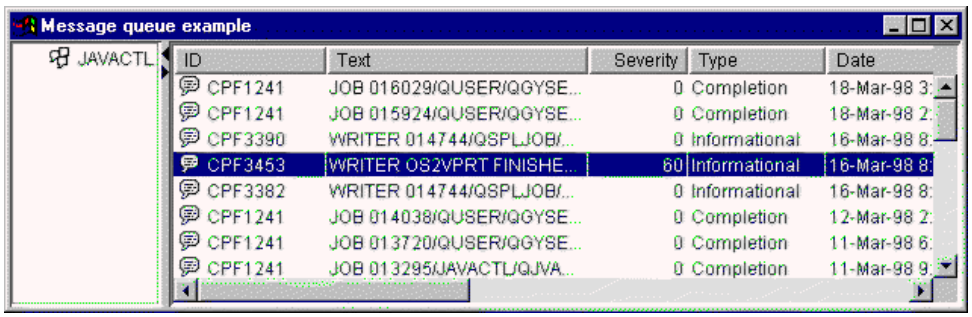
The following example creates a VMessageQueue and presents it in an AS400ExplorerPane:

```
                    // 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);
```

Example

Present the list of messages in a message queue using an AS400ExplorerPane with a VMessageQueue object. Figure 1 shows the VMessageQueue graphical user interface component:

Figure 1: VMessageQueue GUI component



Permission classes

The Permission classes information can be used in a graphical user interface (GUI) through the VIFSFile and VIFSDirectory classes. Permission has been added as an action in each of these classes.

The following example shows how to use Permission with the VIFSDirectory class:

```
// 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 print classes

The following components in the vaccess package allow a Java program to present lists of server print resources in a graphical user interface:

- A VPrinters object is a resource that represents a list of printers for use in AS400Panes.
- A VPrinter object is a resource that represents a printer and its spooled files for use in AS400Panes.
- A VPrinterOutput object is a resource that represents a list of spooled files for use in AS400Panes.
- A SpooledFileViewer object is a resource that visually represents spooled files.

AS400Panes are GUI components that present and allow manipulation of one or more server resources. VPrinters, VPrinter, and VPrinterOutput objects are resources that represent lists of server print resources in AS400Panes.

You can use AS400Pane, VPrinters, VPrinter, and VPrinterOutput objects together to present many views of print resources and to allow the user to select and perform operations on them.

VPrinters class:

A VPrinters object is a resource that represents a list of printers for use in AS400Panes.

To use a VPrinters object, set the system property. Set this property by using a constructor or through the setSystem() method. The VPrinters object is then ″plugged″ into the AS400Pane as the root, using the pane’s constructor or setRoot() method.

A VPrinters object has another useful property for defining the set of printers that is presented in AS400Panes. Use setPrinterFilter() to specify a filter that defines which printers should appear.

When AS400Pane and VPrinters objects are created, they are initialized to a default state. The list of printers has not been loaded. To load the contents, the caller must explicitly call the load() method on either object.

At run-time, a user can perform actions on any printer list or printer by right-clicking it to display the context menu. The printer list context menu can include an item called **Properties** that allows the user to set the printer filter property, which can change the contents of the list.

The printer context menu can include the following items:

- **Hold** - holds the printer

- **Release** - releases the printer
- **Start** - starts the printer
- **Stop** - stops the printer
- **Make available** - makes the printer available
- **Make unavailable** - makes the printer unavailable
- **Properties** - displays properties of the printer and allows the user to set filters

Users can only access printers to which they are authorized. In addition, the caller can prevent the user from performing actions by using the setAllowActions() method on the pane.

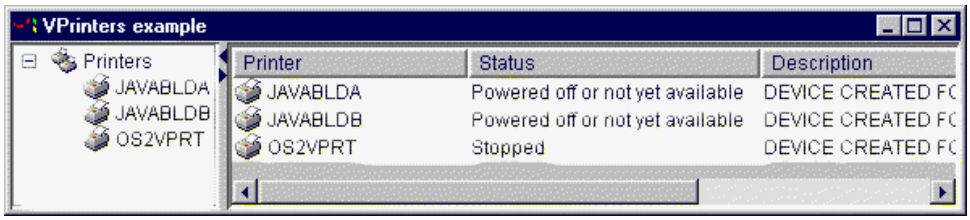
The following example creates a VPrinters object and presents it in an AS400TreePane

```
                    // 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);
```

Example

Present print resources using an AS400ExplorerPane with a VPrinters object. Figure 1 shows the VPrinters graphical user interface component:

Figure 1: VPrinters GUI component



VPrinter class:

A VPrinter object is a resource that represents a server printer and its spooled files for use in AS400Panes.

To use a VPrinter, set the printer property. Set this property by using a constructor or through the setPrinter() method. The VPrinter object is then "plugged" into the AS400Pane as the root, using the pane's constructor or setRoot() method.

When AS400Pane and VPrinter objects are created, they are initialized to a default state. The printer's attributes and list of spooled files are not loaded at creation time.

To load the contents, the caller must explicitly call the load() method on either object. This will initiate communication to the server to gather the contents of the list.

At run-time, a user can perform actions on any printer or spooled file by right-clicking it to display the context menu. The context menu for message queues can include the following items:

- **Hold** - holds the printer
- **Release** - releases the printer
- **Start** - starts the printer
- **Stop** - stops the printer
- **Make available** - makes the printer available
- **Make unavailable** - makes the printer unavailable
- **Properties** - displays properties of the printer and allows the user to set filters

The context menu for spooled files listed for a printer can include the following items:

- **Reply** - replies to the spooled file
- **Hold** - holds the spooled file
- **Release** - releases the spooled file
- **Print next** - prints the next spooled file
- **Send** - sends the spooled file
- **Move** - moves the spooled file
- **Delete** - deletes the spooled file
- **Properties** - displays many properties of the spooled file and allows the user to change some of them

Users can only access printers and spooled files to which they are authorized. In addition, the caller can prevent the user from performing actions by using the setAllowActions() method on the pane.

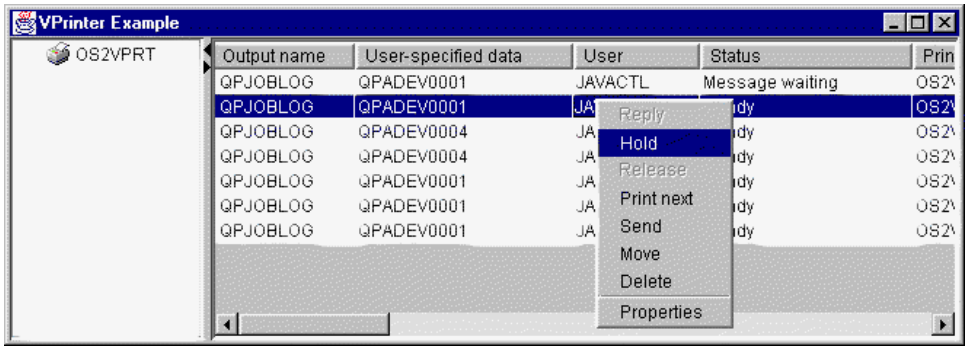
The following example creates a VPrinter and presents it in an AS400ExplorerPane:

```
                    // 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);
```

Example

Present print resources using an AS400ExplorerPane with a VPrinter object. Figure 1 shows the VPrinter graphical user interface component:

Figure 1: VPrinter GUI component



VPrinterOutput class:

A VPrinterOutput object is a resource that represents a list of spooled files on a server for use in AS400Panes.

To use a VPrinterOutput object, set the system property. This property can be set using a constructor or through the setSystem() method. The VPrinterOutput object is then "plugged" into the AS400Pane as the root, using the constructor or setRoot() method of the AS400Pane.

A VPrinterOutput object has other useful properties for defining the set of spooled files that is presented in AS400Panes. Use setFormTypeFilter() to specify which types of forms should appear. Use setUserDataFilter() to specify which user data should appear. Finally, use setUserFilter() to specify which users spooled files should appear.

When AS400Pane and VPrinterOutput objects are created, they are initialized to a default state. The list of spooled files is not loaded at creation time. To load the contents, the caller must explicitly call the load() method on either object. This will initiate communication to the server to gather the contents of the list.

At run-time, a user can perform actions on any spooled file or spooled file list by right-clicking it to display the context menu. The spooled file list context menu can include an item called **Properties** that allows the user to set the filter properties, which can change the contents of the list.

The spooled file context menu can include the following items:

- **Reply** - replies to the spooled file
- **Hold** - holds the spooled file
- **Release** - releases the spooled file
- **Print next** - prints the next spooled file
- **Send** - sends the spooled file
- **Move** - moves the spooled file
- **Delete** - deletes the spooled file
- **Properties** - displays many properties of the spooled file and allows the user to change some of them

Of course, users can only access spooled files to which they are authorized. In addition, the caller can prevent the user from performing actions by using the setAllowActions() method on the pane.

The following example creates a VPrinterOutput and presents it in an AS400ListPane:

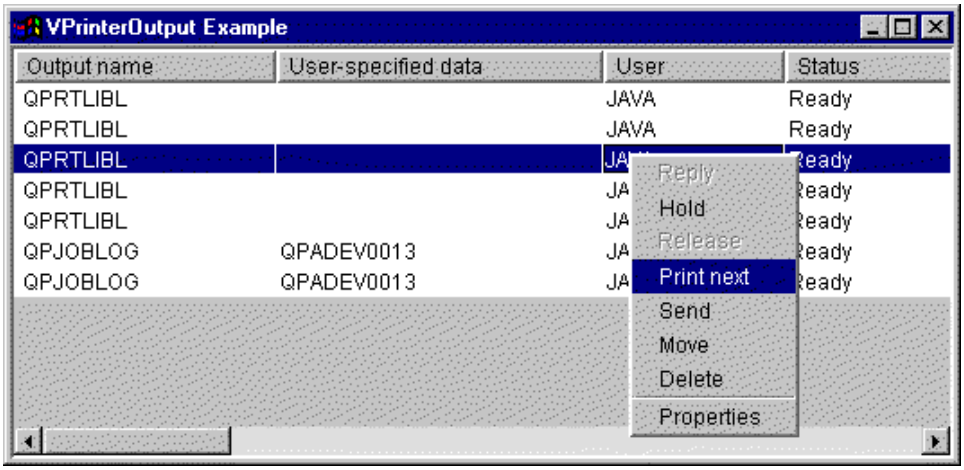
```
                    // Create the VPrinterOutput object.
                    // Assume that "system" is an AS400
                    // object created and initialized
                    // elsewhere.
VPrinterOutput root = new VPrinterOutput (system);

                    // 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);
```

Example

Present a list of spooled files by using the print resource, VPrinterOutput object. Figure 1 shows the VPrinterOutput graphical user interface component:

Figure 1: VPrinterOutput GUI component



SpooledFileViewer class:

The SpooledFileViewer class creates a window for viewing Advanced Function Printing (AFP) and Systems Network Architecture character string (SCS) files that have been spooled for printing. The class essentially adds a "print preview" function to your spooled files, common to most word processing programs, as illustrated in Figure 1.

The spooled file viewer is especially helpful when viewing the accuracy of the layout of the files is more important than printing the files, or when viewing the data is more economical than printing, or when a printer is not available.

Note: SS1 Option 8 (AFP Compatibility Fonts) must be installed on the host server.

Using the SpooledFileViewer class

Three constructor methods are available to create an instance of the SpooledFileViewer class. The SpooledFileViewer() constructor can be used to create a viewer without a spooled file associated with it. If this constructor is used, a spooled file will need to be set later using setSpooledFile(SpooledFile). The SpooledFileViewer(SpooledFile) constructor can be used to create a viewer for the given spooled file, with page one as the initial view. Finally, the SpooledFileViewer(spooledFile, int) constructor can be used to create a viewer for the given spooled file with the specified page as the initial view. No matter which constructor is used, once a viewer is created, a call to load() must be performed in order to actually retrieve the spooled file data.

Then, your program can traverse the individual pages of the spooled file by using the following methods:

- load FlashPage()
- load Page()
- pageBack()
- pageForward()

If, however, you need to examine particular sections of the document more closely, you can magnify or reduce the image of a page of the document by altering the ratio proportions of each page with the following:

- fitHeight()
- fitPage()
- fitWidth()
- actualSize()

Your program concludes with calling the close() method that closes the input stream and releases any resource associations with the stream.

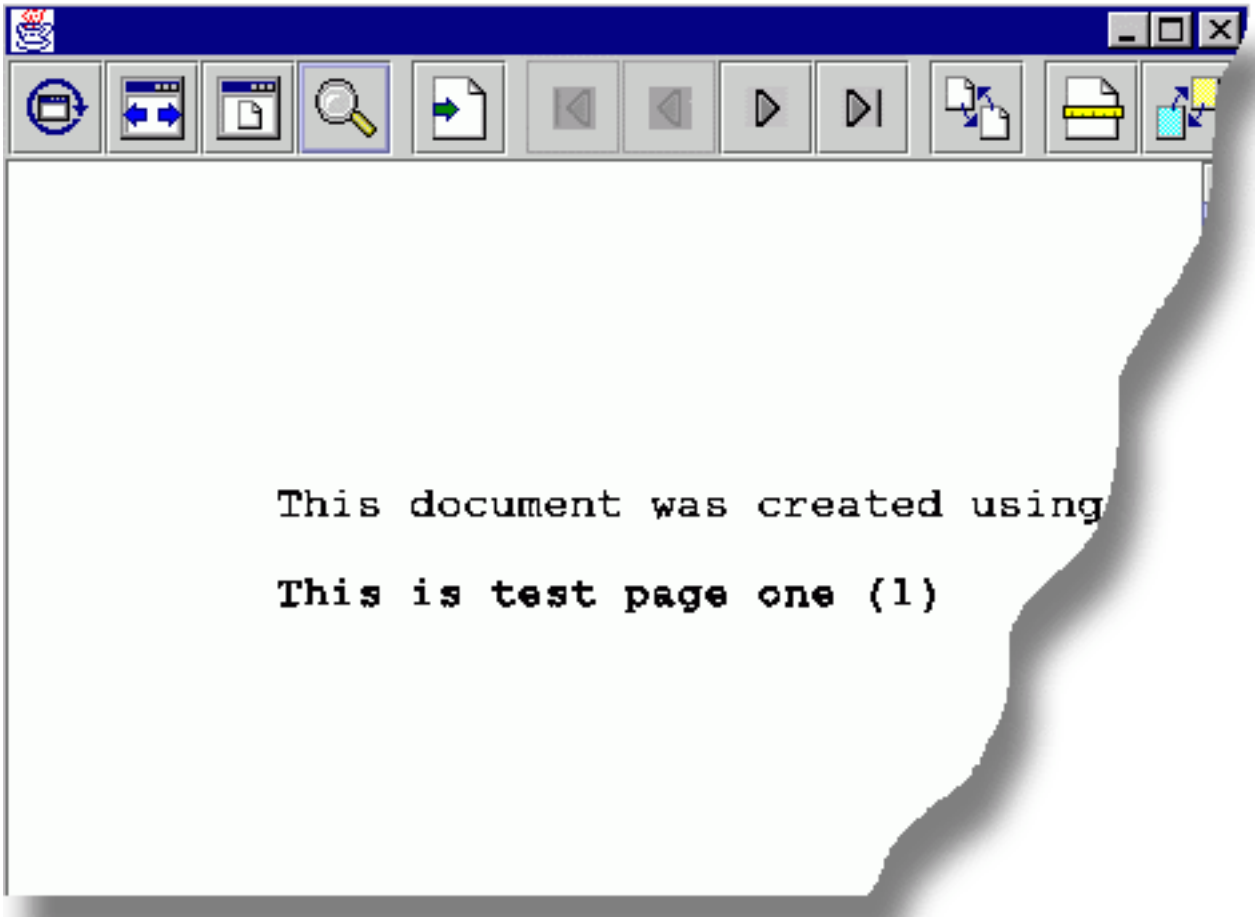
Using the SpooledFileViewer

An instance of the SpooledFileViewer class is actually a graphical representation of a viewer capable of displaying and navigating through an AFP or SCS spooled file. For example, the following code creates the spooled file viewer in Figure 1 to display a spooled file previously created on the server.

Note: You can either select a button on the image in Figure 1 for an explanation of its function, or (if your browser is not JavaScript™ enabled) see the toolbar description.

```
// 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();
```

Figure 1: SpooledFileViewer



SpooledFileViewer Toolbar description


The actual size button returns the spooled file page image to its original size by using the actualSize() method.


The fit width button stretches the spooled file page image to the left and right edges of the viewer's frame by using the fitWidth() method.


The fit page button stretches the spooled file page image vertically and horizontally to fit within the spooled file viewer's frame by using the fitPage() method.


The zoom button allows you to increase or decrease the size of the spooled file page image by selecting one of the preset percentages or entering your own percent in a text field that appears in a dialog box after selecting the zoom button.


The go to page button allows you to go to a specific page within the spooled file when selected.

 The first page button takes you to the first page of the spooled file when selected and indicates that you are on the first page when deactivated.

 The previous page button takes you to the page immediately before the page you are viewing when selected.

 The next page button advances you to the page immediately after the page you are viewing when selected.

 The last page button advances you to the last page of the spooled file when selected and indicates that you are on the last page when deactivated.

 The load flash page button loads the previously viewed page by using the loadFlashPage() method when selected.

 The set paper size button allows you to set the paper size when selected.

 The set viewing fidelity button allows you to set the viewing fidelity when selected.

Vaccess ProgramCall classes

The program call components in the vaccess package allow a Java program to present a button or menu item that calls a server program. Input, output, and input/output parameters can be specified using ProgramParameter objects. When the program runs, the output and input/output parameters contain data returned by the server program.

A ProgramCallButton object represents a button that calls an server program when pressed. The ProgramCallButton class extends the Java Foundation Classes (JFC) JButton class so that all buttons have a consistent appearance and behavior.

Similarly, a ProgramCallMenuItem object represents a menu item that calls an server program when selected. The ProgramCallMenuItem class extends the JFC JMenuItem class so that all menu items also have a consistent appearance and behavior.

To use a vaccess program call component, set both the system and program properties. Set these properties by using a constructor or through the setSystem() and setProgram() methods.

The following example creates a ProgramCallMenuItem. At run time, when the menu item is selected, it calls a program:

```
// 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);
```

When a server program runs, it may return zero or more server messages. To detect when the server program runs, add an ActionCompletedListener to the button or menu item using the addActionCompletedListener() method. When the program runs, it fires an ActionCompletedEvent to all such listeners. A listener can use the getMessageList() method to retrieve any server messages that the program generated.

This example adds an ActionCompletedListener that processes all server messages that the program generated:

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

Parameters

ProgramParameter objects are used to pass parameter data between the Java program and the server program. Input data is set with the setInputData() method. After the program is run, output data is retrieved with the getOutputData() method.

Each parameter is a byte array. It is up to the Java program to convert the byte array between Java and server formats. The data conversion classes provide methods for converting data.

You can add parameters to a program call graphical user interface component one at a time using the addParameter() method or all at once using the setParameterList() method.

For more information about using ProgramParameter objects, see the ProgramCall access class.

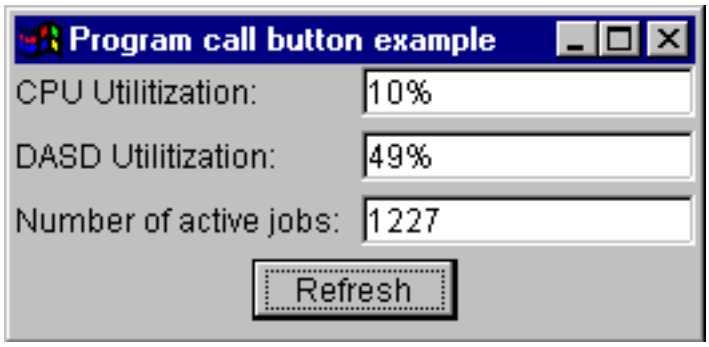
The following example adds two parameters:

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

Examples

Example of using a ProgramCallButton in an application. Figure 1 shows how the ProgramCallButton looks:

Figure 1: Using ProgramCallButton in an application



Vaccess record-level access classes

The record-level access classes in the vaccess package allow a Java program to present various views of server files.

The following components are available:

- RecordListFormPane presents a list of records from a server file in a form.
- RecordListTablePane presents a list of records from a server file in a table.
- RecordListTableModel manages the list of records from a server file for a table.

Keyed access

You can use the record-level access graphical user interface components with keyed access to a server file. Keyed access means that the Java program can access the records of a file by specifying a key.

Keyed access works the same for each record-level access graphical user interface component. Use setKeyed() to specify keyed access instead of sequential access. Specify a key using the constructor or the setKey() method. See Specifying the key for more information about how to specify the key.

By default, only records whose keys are equal to the specified key are displayed. To change this, specify the searchType property using the constructor or setSearchType() method. Possible choices are as follows:

- KEY_EQ - Display records whose keys are equal to the specified key.
- KEY_GE - Display records whose keys are greater than or equal to the specified key.
- KEY_GT - Display records whose keys are greater than the specified key.
- KEY_LE - Display records whose keys are less than or equal to the specified key.
- KEY_LT - Display records whose keys are less than the specified key.

The following example creates a RecordListTablePane object to display all records less than or equal to a key.

```
                    // 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 class:

A RecordListFormPane presents the contents of a server file in a form. The form displays one record at a time and provides buttons that allow the user to scroll forward, backward, to the first or last record, or refresh the view of the file contents.

To use a RecordListFormPane, set the system and fileName properties. Set these properties by using the constructor or the setSystem() and setFileName() methods. Use load() to retrieve the file contents and present the first record. When the file contents are no longer needed, call close() to ensure that the file is closed.

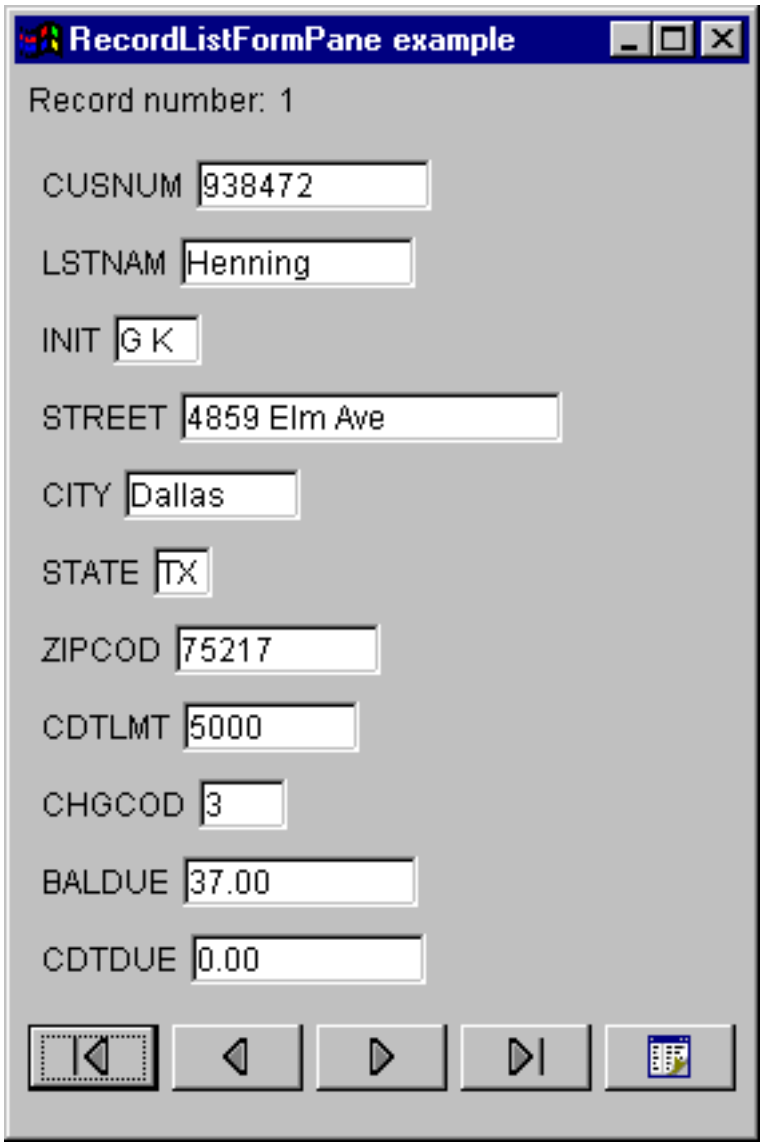
The following example creates a RecordListFormPane object and adds it to a frame:

```
                    // 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);
```

Example

Present an RecordListFormPane which displays the contents of a file. Figure 1 shows the RecordListFormPane graphical user interface component:

Figure 1: RecordListFormPane GUI component



RecordListTablePane class:

A RecordListTablePane presents the contents of a server file in a table. Each row in the table displays a record from the file and each column displays a field.

To use a RecordListTablePane, set the system and fileName properties. Set these properties by using the constructor or the setSystem() and setFileName() methods. Use load() to retrieve the file contents and present the records in the table. When the file contents are no longer needed, call close() to ensure that the file is closed.

The following example creates a RecordListTablePane object and adds it to a frame:

```
// Create an RecordListTablePane
// object. Assume that "system" is
// an AS400 object that is created
// and initialized elsewhere.
```

```
RecordListTablePane tablePane = new RecordListTablePane (system,
                                  "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");
                    // Load the file contents.
tablePane.load ();

                    // Add the table pane to a frame.
                    // Assume that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (tablePane);
```

RecordListTablePane and RecordListTableModel classes:

RecordListTablePane is implemented using the model-view-controller paradigm, in which the data and the user interface are separated into different classes. The implementation integrates RecordListTableModel with Java Foundation Classes' (JFC) JTable. The RecordListTableModel class retrieves and manages the contents of the file and JTable displays the file contents graphically and handles user interaction.

RecordListTablePane provides enough functionality for most requirements. However, if a caller needs more control of the JFC component, then the caller can use RecordListTableModel directly and provide customized integration with a different graphical user interface component.

To use a RecordListTableModel, set the system and fileName properties. Set these properties by using the constructor or the setSystem() and setFileName() methods. Use load() to retrieve the file contents. When the file contents are no longer needed, call close() to ensure that the file is closed.

The following example creates a RecordListTableModel object and presents it with a JTable:

```
                    // Create a RecordListTableModel
                    // object. Assume that "system" is
                    // an AS400 object that is created
                    // and initialized elsewhere.
RecordListTableModel tableModel = new RecordListTableModel (system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                    // Load the file contents.
tableModel.load ();

                    // Create a JTable for the model.
JTable table = new JTable (tableModel);

                    // Add the table to a frame. Assume
                    // that "frame" is a JFrame
                    // created elsewhere.
frame.getContentPane ().add (table);
```

ResourceListPane and ResourceListDetailsPane

Use the ResourceListPane and ResourceListDetailsPane classes to present a resource list in a graphical user interface (GUI).

- ResourceListPane displays the contents of the resource list in a graphical javax.swing.JList. Every item displayed in the list represents a resource object from the resource list.
- ResourceListDetailsPane displays the contents of the resource list in a graphical javax.swing.JTable. Every row in the table represents a resource object from the resource list.

The table columns for a ResourceListDetailsPane are specified as an array of column attribute IDs. The table contains a column for each element of the array and a row for each resource object.

Pop-up menus are enabled by default for both ResourceListPane and ResourceListDetailsPane.

Most errors are reported as com.ibm.as400.vaccess.ErrorEvents rather than thrown exceptions. Listen for ErrorEvents in order to diagnose and recover from error conditions.

Example: Displaying a resource list in a GUI

This example creates a ResourceList of all users on a system and displays it in a GUI (details pane):

```
// Create the resource list.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RUserList userList = new RUserList(system);

// Create the ResourceListDetailsPane.  In this example,
// there are two columns in the table.  The first column
// contains the icons and names for each user.  The
// second column contains the text description for each
// user.
Object[] columnAttributeIDs = new Object[] { null, RUser.TEXT_DESCRIPTION };
ResourceListDetailsPane detailsPane = new ResourceListDetailsPane();
detailsPane.setResourceList(userList);
detailsPane.setColumnAttributeIDs(columnAttributeIDs);

// Add the ResourceListDetailsPane to a JFrame and show it.
JFrame frame = new JFrame("My Window");
frame.getContentPane().add(detailsPane);
frame.pack();
frame.show();

// The ResourceListDetailsPane will appear empty until
// we load it.  This gives us control of when the list
// of users is retrieved from the iSeries.
detailsPane.load();
```

System status classes

The System Status components in the vaccess package allow you to create GUIs by using the existing AS400Panes. You also have the option to create your own GUIs using the Java Foundation Classes (JFC). The VSystemStatus object represents a system status on the server. The VSystemPool object represents a system pool on the server. The VSystemStatusPane represents a visual pane that displays the system status information.

The VSystemStatus class allows you to get information about the status of a server session within a GUI environment:

- The getSystem() method returns the server where the system status information is contained
- The getText() method returns the description text
- The setSystem() method sets the server where the system status information is located

In addition to the methods mentioned above, you can also access and change system pool information in a GUI.

You use VSystemStatus with VSystemStatusPane. VSystemPane is the visual display pane where information is shown for both system status and system pool.

VSystemPool class:

The VSystemPool class allows you to retrieve and set system pool information from a server using a GUI design. VSystemPool works with various panes in the vaccess package including the VSystemStatusPane.

The following list is some of the methods that are available to use in VSystemPool:

- The getActions() method returns a list of actions that you can perform
- The getSystem() method returns the server where the system pool information is found
- The setSystemPool() method sets the system pool object

VSystemStatusPane class:

The VSystemStatusPane class allows a Java program to display system status and system pool information.

VSystemStatusPane includes the following methods:
- getVSystemStatus(): Returns the VSystemStatus information in a VSystemStatusPane.
- setAllowModifyAllPools(): Sets the value to determine if system pool information can be modified.

The following example shows you how to use the VSystemStatusPane class:

```
// 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();
```

System values GUI

The system value components in the vaccess package allow a Java program to create GUIs by using the existing AS400Panes or by creating your own panes using the Java Foundation Classes(JFC). The VSystemValueList object represents a system value list on the server.

To use the System Value GUI component, set the system name with a constructor or through the setSystem() method.

Example The following example creates a system value GUI using the AS400Explorer Pane:

```
//Create an AS400 object
AS400 mySystem = newAS400("mySystem.myCompany.com");
VSystemValueList mySystemValueList = new VSystemValueList(mySystem);
as400Panel=new AS400ExplorerPane((VNode)mySystemValueList);
//Create and load an AS400ExplorerPane object
as400Panel.load();
```

Vaccess users and groups classes

The users and groups components in the vaccess package allow you to present lists of server users and groups through the VUser class.

The following components are available:
- AS400Panes are GUI components that present and allow manipulation of one or more server resources.
- A VUserList object is a resource that represents a list of server users and groups for use in AS400Panes.
- A VUserAndGroup object is a resource for use in AS400Panes that represents groups of server users. It allows a Java program to list all users, list all groups, or list users who are not in groups.

AS400Pane and VUserList objects can be used together to present many views of the list. They can also be used to allow the user to select users and groups.

To use a VUserList, you must first set the system property. Set this property by using a constructor or through the setSystem() method. The VUserList object is then ″plugged″ into the AS400Pane as the root, using the constructor or setRoot() method of the AS400Pane.

VUserList has some other useful properties for defining the set of users and groups that are presented in AS400Panes:

- Use the setUserInfo() method to specify the types of users that should appear.
- Use the setGroupInfo() method to specify a group name.

You can use the VUserAndGroup object to get information about the Users and Groups on the system. Before you can get information about a particular object, you need to load the information so that it can be accessed. You can display the server in which the information is found by using the getSystem method.

When AS400Pane objects and VUserList or VUserAndGroup objects are created, they are initialized to a default state. The list of users and groups has not been loaded. To load the contents, the Java program must explicitly call the load() method on either object to initiate communication to the server to gather the contents of the list.

At run-time, right-click a user, user list, or group to display the shortcut menu. Select **Properties** from the shortcut menu to perform actions on the selected object:

- User - Display a list of user information including the description, user class, status, job description, output information, message information, international information, security information, and group information.
- User list - Work with user information and group information properties. You can also change the contents of the list.
- Users and groups - Display properties, such as the user name and description.

Users can only access users and groups to which they are authorized. In addition, the Java program can prevent the user from performing actions by using the setAllowActions() method on the pane.

The following example creates a VUserList and presents it in an AS400DetailsPane:

```
// 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);
```

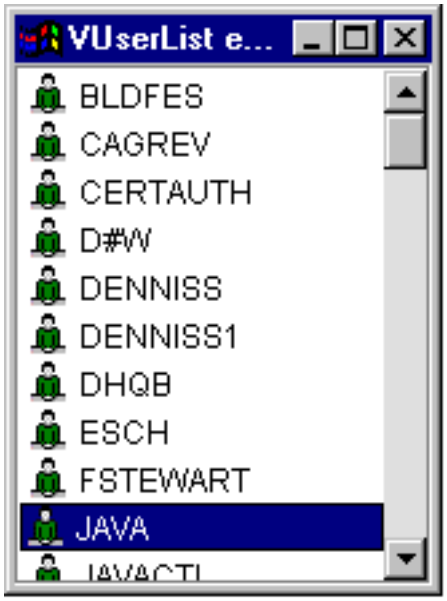
The following example shows how to use the VUserAndGroup object:

```
// 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);
```

Other Examples

Present a list of users on the system using an AS400ListPane with a VUserList object.

The following image shows the VUserList graphical user interface component:



Graphical Toolbox and PDML

The Graphical Toolbox, a set of UI tools, makes it easy to create custom user interface panels in Java.

You can incorporate the panels into your Java applications, applets, or iSeries Navigator plug-ins. The panels may contain data obtained from the iSeries, or data obtained from another source such as a file in the local file system or a program on the network.

The **GUI Builder** is a WYSIWYG visual editor for creating Java dialogs, property sheets and wizards. With the GUI Builder you can add, arrange, or edit user interface controls on a panel, and then preview the panel to verify the layout behaves the way you expected. The panel definitions you create can be used in dialogs, inserted within property sheets and wizards, or arranged into splitter, deck, and tabbed panes. The GUI Builder also allows you to build menu bars, toolbars, and context menu definitions. You can also incorporate JavaHelp in your panels, including context sensitive help.

The **Resource Script Converter** converts Windows resource scripts into an XML representation that is usable by Java programs. With the Resource Script Converter you can process Windows resource scripts (RC files) from your existing Windows dialogs and menus. These converted files can then be edited with the GUI Builder. Property sheets and wizards can be made from RC files using the resource script converter along with the GUI Builder.

Underlying these two tools is a new technology called the **Panel Definition Markup Language**, or **PDML**. PDML is based on the Extensible Markup Language (XML) and defines a platform-independent language for describing the layout of user interface elements. Once your panels are defined in PDML, you can use the runtime API provided by the Graphical Toolbox to display them. The API displays your panels by interpreting the PDML and rendering your user interface using the Java Foundation Classes.

Note: Using PDML requires that you run version 1.4 or later of the Java Runtime Environment.

Benefits of the Graphical Toolbox

Write Less Code and Save Time

With the Graphical Toolbox you have the ability to create Java-based user interfaces quickly and easily. The GUI Builder lets you have precise control over the layout of UI elements on your

panels. Because the layout is described in PDML, you are not required to develop any Java code to define the user interface, and you do not need to recompile code in order to make changes. As a result, significantly less time is required to create and maintain your Java applications. The Resource Script Converter lets you migrate large numbers of Windows panels to Java quickly and easily.

Custom Help

Defining user interfaces in PDML creates some additional benefits. Because all of a panel's information is consolidated in a formal markup language, the tools can be enhanced to perform additional services on behalf of the developer. For example, both the GUI Builder and the Resource Script Converter are capable of generating HTML skeletons for the panel's online help. You decide which help topics are required and the help topics are automatically built based on your requirements. Anchor tags for the help topics are built right into the help skeleton, which frees the help writer to focus on developing appropriate content. The Graphical Toolbox runtime environment automatically displays the correct help topic in response to a user's request.

Automatic Panel to Code Integration

In addition, PDML provides tags that associate each control on a panel with an attribute on a JavaBean. Once you have identified the bean classes that will supply data to the panel and have associated a attribute with each of the appropriate controls, you can request that the tools generate Java source code skeletons for the bean objects. At runtime, the Graphical Toolbox automatically transfers data between the beans and the controls on the panel that you identified.

Platform Independent

The Graphical Toolbox runtime environment provides support for event handling, user data validation, and common types of interaction among the elements of a panel. The correct platform look and feel for your user interface is automatically set based on the underlying operating system, and the GUI Builder lets you toggle the look and feel so that you can evaluate how your panels will look on different platforms.

The Graphical Toolbox provides you with two tools and, therefore, two ways of automating the creation of your user interfaces. You can use the GUI Builder to quickly and easily create new panels from scratch, or you can use the Resource Script Converter to convert existing Windows-based panels to Java. The converted files can then be edited with GUI Builder. Both tools support internationalization.

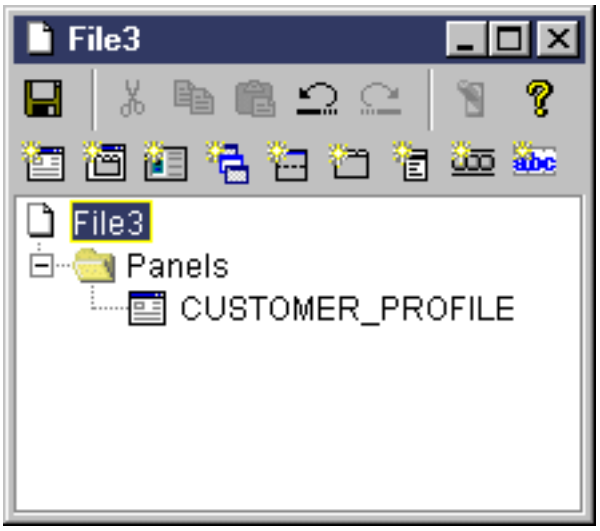
GUI Builder

Two windows are displayed when you invoke the GUI Builder for the first time, as shown in Figure 1:

Figure 1: GUI Builder windows

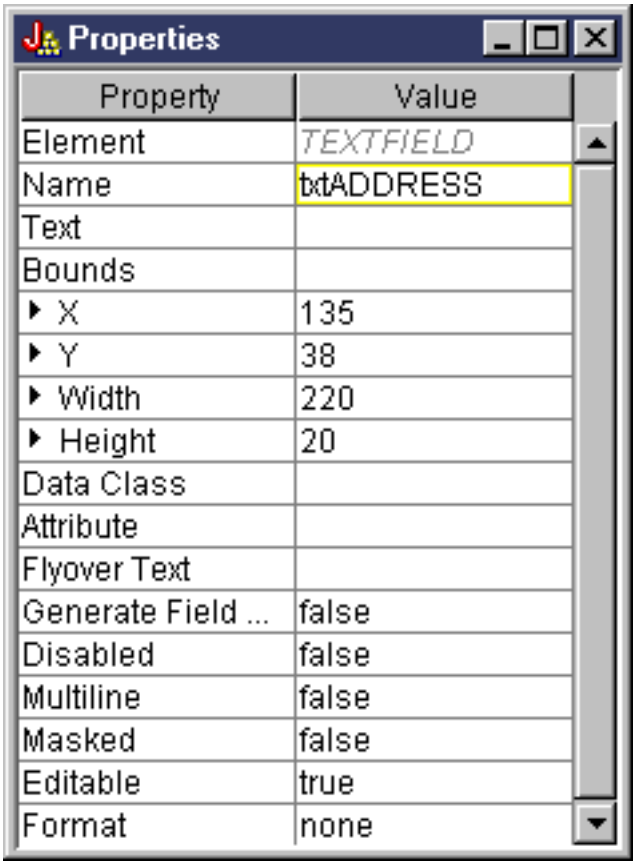
Use the File Builder window to create and edit your PDML files.

Figure 2: File Builder window



Use the Properties window to view or change the properties of the currently selected control.

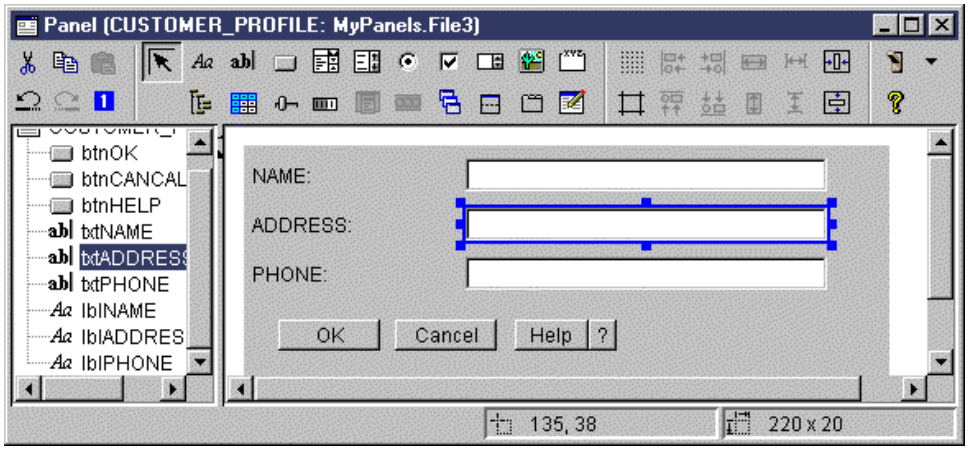
Figure 3: Properties window



Use the Panel Builder window to create and edit your graphical user interface components. Select the desired component from the toolbar and click on the panel to place it where ever you want. The toolbar

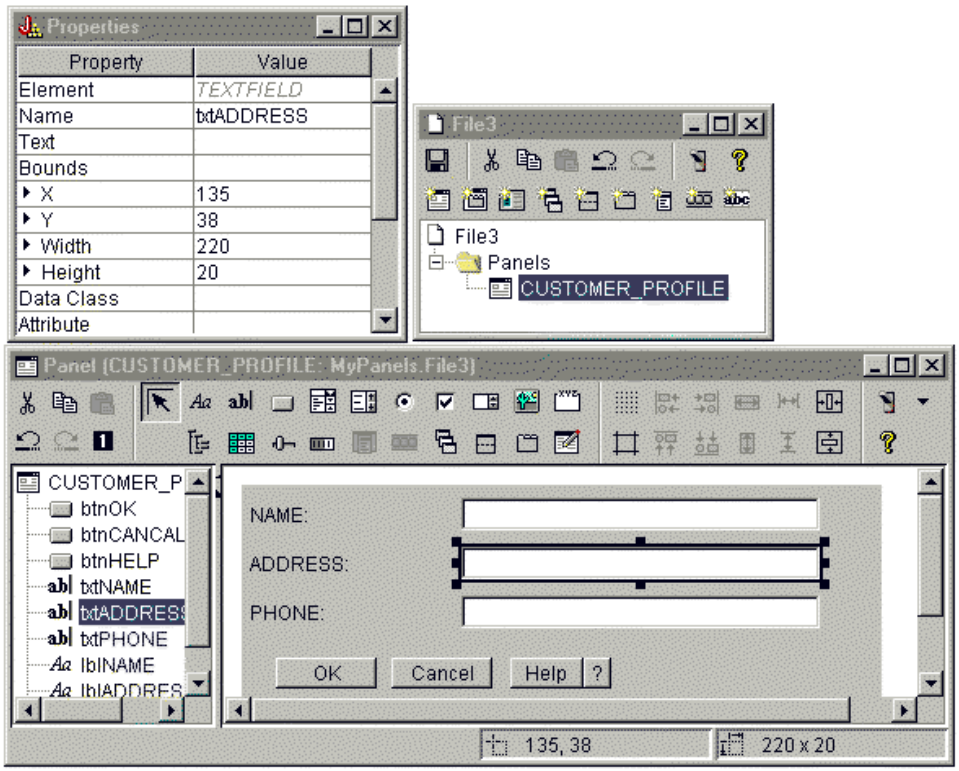
also facilities for aligning groups of controls, for previewing the panel, and for requesting online help for a GUI Builder function. See GUI Builder Panel Builder toolbar for a description of what each icon does.

Figure 4: Panel Builder window



The panel being edited is displayed in the Panel Builder window. Figure 5 shows how the windows work together:

Figure 5: Example of how GUI Builder windows work together

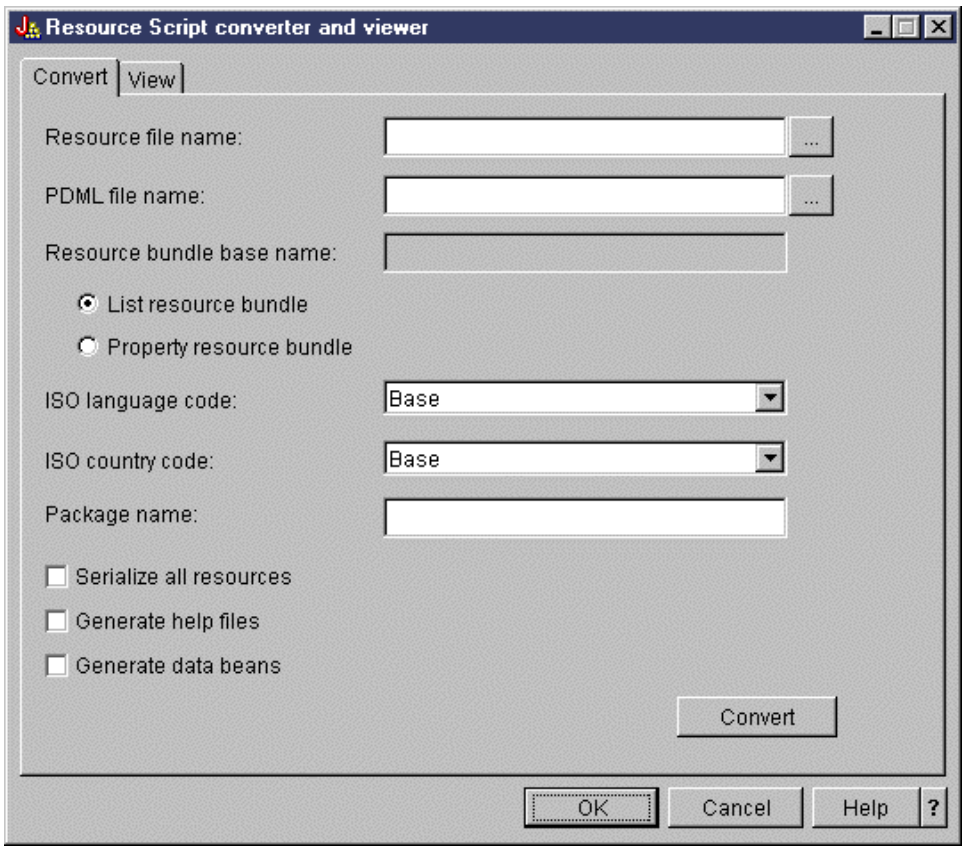


Resource Script Converter

The Resource Script Converter consists of a two-paned tabbed dialog. On the **Convert** pane you specify the name of the Microsoft or VisualAge® for Windows RC file that is to be converted to PDML. You can specify the name of the target PDML file and associated Java resource bundle that will contain the

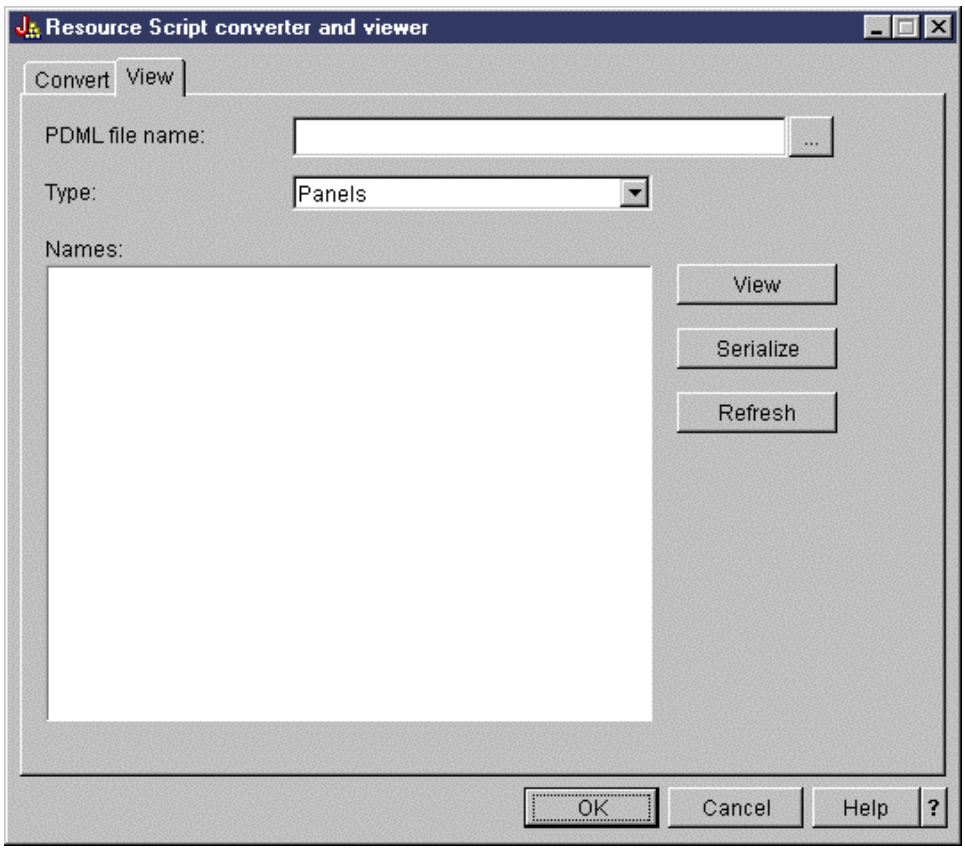
translated strings for the panels. In addition, you can request that online help skeletons be generated for the panels, generate Java source code skeletons for the objects that supply data to the panels, and serialize the panel definitions for improved performance at runtime. The Converter's online help provides a detailed description of each input field on the Convert pane.

Figure 6: Resource Script Converter Convert pane



After the conversion has run successfully, you can use the **View** pane to view the contents of your newly-created PDML file, and preview your new Java panels. You can use the GUI Builder to make minor adjustments to a panel if needed. The Converter always checks for an existing PDML file before performing a conversion, and attempts to preserve any changes in case you need to run the conversion again later.

Figure 7: Resource Script Converter View pane



Setting up the Graphical Toolbox

The Graphical Toolbox is delivered as a set of JAR files. To set up the Graphical Toolbox you must install the JAR files on your workstation and set your CLASSPATH environment variable.

You must also ensure that your workstation meets the requirements to run IBM Toolbox for Java.

Installing the Graphical Toolbox on your workstation

To develop Java programs using the Graphical Toolbox, first install the Graphical Toolbox JAR files on your workstation. Use one of the following methods:

Transfer the JAR Files

Note: The following list represents some of the methods you can use to transfer the JAR files. The IBM Toolbox for Java licensed program must be installed on your iSeries. Additionally, you need to download the JAR file for JavaHelp, jhall.jar, from the Sun JavaHelp Web site.

- Use FTP (ensure you transfer the files in binary mode) and copy the JAR files from the directory **/QIBM/ProdData/HTTP/Public/jt400/lib** to a local directory on your workstation
- Use iSeries Access for Windows to map a network drive.

Install JAR files with iSeries Access for Windows

You can also install the Graphical Toolbox when you install iSeries Access for Windows. The IBM Toolbox for Java is now shipped as part of iSeries Access for Windows. If you are installing iSeries Access for Windows for the first time, choose Custom Install and select the **IBM Toolbox for Java** component on the install menu. If you have already installed iSeries Access for Windows, you can use the Selective Setup program to install this component if it is not already present.

Setting your classpath

To use the Graphical Toolbox, you must add these JAR files to your CLASSPATH environment variable (or specify them on the `classpath` option on the command line).

For example, if you have copied the files to the directory **C:\gtbox\lib** on your workstation, you must add the following path names to your classpath:

```
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;
```

You also need to add an XML parser to your CLASSPATH. For more information, see the following page:

"XML parser and XSLT processor" on page 393

If you have installed the Graphical Toolbox using iSeries Access for Windows, the JAR files (except jhall.jar) will all reside in the directory **\Program Files\Ibm\Client Access\jt400\lib** on the drive where you have installed iSeries Access for Windows. iSeries Access for Windows installs jhall.jar in the **\Program Files\Ibm\Client Access\jre\lib** directory. The path names in your classpath reflect this.

JAR File Descriptions

- **uitools.jar**: Contains the GUI Builder and Resource Script Converter tools.
- **jui400.jar**: Contains the runtime API for the Graphical Toolbox. Java programs use this API to display the panels constructed using the tools. These classes may be redistributed with applications.
- **data400.jar**: Contains the runtime API for the Program Call Markup Language (PCML). Java programs use this API to call iSeries programs whose parameters and return values are identified using PCML. These classes may be redistributed with applications.
- **util400.jar**: Contains utility classes for formatting iSeries data and handling iSeries messages. These classes may be redistributed with applications.
- **jhall.jar**: Contains the JavaHelp classes that displays the online help and context sensitive help for the panels you build with the GUI Builder.
- **XML parser**: Contains the XML parser used by the API classes to interpret PDML and PCML documents.

Note: You can use internationalized versions of the GUI Builder and Resource Script Converter tools. To run a non-U.S. English version, you must add to your Graphical Toolbox installation the correct version of **uitools.jar** for your language and country or region. These JAR files are available on the iSeries server in **/QIBM/ProdData/HTTP/Public/jt400/Mri29xx**, where 29xx is the 4-digit i5/OS NLV code that corresponds to your language and country or region. (The names of the JAR files in the various Mri29xx directories include 2-character suffixes for the Java language code and the country or region code.) This additional JAR file is be added to your classpath ahead of **uitools.jar** in the search order.

Using the Graphical Toolbox

Once you have installed the Graphical Toolbox, follow these links to learn how to use the tools:

- Using the GUI Builder
- Using the Resource Script Converter

Creating your user interface

You can create your user interface using the GUI Builder tool.

To start the GUI Builder, use the following command:

```
java com.ibm.as400.ui.tools.GUIBuilder [-plaf look and feel]
```

If you did not set your CLASSPATH environment variable to contain the Graphical Toolbox JAR files, then you will need to specify them on the command line using the `classpath` option. See Setting Up the Graphical Toolbox.

Options `-plaf` *look and feel*

The platform look and feel that you want. This option lets you override the default look and feel that is set based on the platform you are developing on, so you can preview your panels to see how they will look on different operating system platforms. The following look and feel values are accepted:

- Windows
- Metal
- Motif

Currently, additional look and feel attributes that Swing 1.1 may support are not supported by the GUI Builder

Types of user interface resources

When you start the GUI Builder for the first time, you need to create a new PDML file. From the menu bar on the GUI Builder widow, select **File** --> **New File**. After you create your new PDML file, you can define any of the following types of UI resources that you want it to contain.

Panel The fundamental resource type. It describes a rectangular area within which UI elements are arranged. The UI elements may consist of simple controls, such as radio buttons or text fields, images, animations, custom controls, or more sophisticated subpanels (see the following definitions for Split Pane, Deck Pane and Tabbed Pane). A panel may define the layout for a stand-alone window or dialog, or it may define one of the subpanels that is contained in another UI resource.

Menu A popup window containing one or more selectable actions, each represented by a text string (″Cut″, ″Copy″ and ″Paste″ are examples). You can define mnemonics and accelerator keys for each action, insert separators and cascading submenus, or define special checked or radio button menu items. A menu resource may be used as a stand-alone context menu, as a drop-down menu in a menu bar, or it may itself define the menu bar associated with a panel resource.

Toolbar

A window consisting of a series of push buttons, each representing a possible user action. Each button may contain text, an icon or both. You can define the toolbar as floatable, which lets the user drag the toolbar out of a panel and into a stand-alone window.

Property Sheet

A stand-alone window or dialog consisting of a tabbed panels and OK, Cancel, and Help buttons. Panel resources define the layout of each tabbed window.

Wizard

A stand-alone window or dialog consisting of a series of panels that are displayed to the user in a predefined sequence, with Back, Next, Cancel, Finish, and Help buttons. The wizard window may also display a list of tasks to the left of the panels which track the user's progress through the wizard.

Split Pane

A subpane consisting of two panels separated by a splitter bar. The panels may be arranged horizontally or vertically.

Tabbed Pane
A subpane that forms a tabbed control. This tabbed control can be placed inside of another panel, split pane, or deck pane.

Deck Pane
A subpane consisting of a collection of panels. Of these, only one panel can be displayed at a time. For example, at runtime the deck pane might change the panel which is displayed depending on a given user action.

String Table
A collection of string resources and their associated resource identifiers.

Generated files

The translatable strings for a panel are not stored in the PDML file itself, but in a separate Java resource bundle. The tools let you specify how the resource bundle is defined, either as a Java PROPERTIES file or as a ListResourceBundle subclass. A ListResourceBundle subclass is a compiled version of the translatable resources, which enhances the performance of your Java application. However, it will slow down the GUI Builder's saving process, because the ListResourceBundle will be compiled in each save operation. Therefore it's best to start with a PROPERTIES file (the default setting) until you're satisfied with the design of your user interface.

You can use the tools to generate HTML skeletons for each panel in the PDML file. At runtime, the correct help topic is displayed when the user clicks on the panel's Help button or presses F1 while the focus is on one of the panel's controls. You must insert your help content at the appropriate points in the HTML, within the scope of the <!-- `HELPDOC:SEGMENTBEGIN` --> and <!-- `HELPDOC:SEGMENTEND` --> tags. For more specific help information see Editing Help Documents generated by GUI builder.

You can generate source code skeletons for the JavaBeans™ that will supply the data for a panel. Use the Properties window of the GUI Builder to fill in the DATACLASS and ATTRIBUTE properties for the controls which will contain data. The DATACLASS property identifies the class name of the bean, and the ATTRIBUTE property specifies the name of the gettor/settor methods that the bean class implements. Once you've added this information to the PDML file, you can use the GUI Builder to generate Java source code skeletons and compile them. At runtime, the appropriate gettor/settor methods will be called to fill in the data for the panel.

Note: The number and type of gettor/settor methods is dependent on the type of UI control with which the methods are associated. The method protocols for each control are documented in the class description for the DataBean class.

Finally, you can serialize the contents of your PDML file. Serialization produces a compact binary representation of all of the UI resources in the file. This greatly improves the performance of your user interface, because the PDML file must not be interpreted in order to display your panels.

To summarize: If you have created a PDML file named **MyPanels.pdml**, the following files will also be produced based on the options you have selected on the tools:

- **MyPanels.properties** if you have defined the resource bundle as a PROPERTIES file
- **MyPanels.java** and **MyPanels.class** if you have defined the resource bundle as a ListResourceBundle subclass
- **<panel name>.html** for each panel in the PDML file, if you have elected to generate online help skeletons
- **<dataclass name>.java** and **<dataclass name>.class** for each unique bean class that you have specified on your DATACLASS properties, if you have elected to generate source code skeletons for your JavaBeans
- **<resource name>.pdml.ser** for each UI resource defined in the PDML file, if you've elected to serialize its contents.

Note: The conditional behavior functions (SELECTED/DESELECTED) will not work if the panel name is the same as the one in which the conditional behavior function is being attached. For instance, if PANEL1 in FILE1 has a conditional behavior reference attached to a field that references a field in PANEL1 in FILE2, the conditional behavior event will not work. To fix this, rename PANEL1 in FILE2 and then update the conditional behavior event in FILE1 to reflect this change.

Running the Resource Script Converter

To start the Resource Script Converter, invoke the Java interpreter as follows:

```
java com.ibm.as400.ui.tools.PDMLViewer
```

If you did not set your CLASSPATH environment variable to contain the Graphical Toolbox JAR files, then you will need to specify them on the command line using the `classpath` option. See Setting Up the Graphical Toolbox.

You can also run the Resource Script Converter in batch mode using the following command:

```
java com.ibm.as400.ui.tools.RC2XML file [options]
```

Where *file* is the name of the resource script (RC file) to be processed. **Options**

-x name
The name of the generated PDML file. Defaults to the name of the RC file to be processed.

-p name
The name of the generated PROPERTIES file. Defaults to the name of the PDML file.

-r name
The name of the generated ListResourceBundle subclass. Defaults to the name of the PDML file.

-package name
The name of the package to which the generated resources will be assigned. If not specified, no package statements will be generated.

-l locale
The locale in which to produce the generated resources. If a locale is specified, the appropriate 2-character ISO language and country or region codes are suffixed to the name of the generated resource bundle.

-h Generate HTML skeletons for online help.

-d Generate source code skeletons for JavaBeans.

-s Serialize all resources.

Mapping Windows Resources to PDML

All dialogs, menus, and string tables found in the RC file will be converted to the corresponding Graphical Toolbox resources in the generated PDML file. You can also define DATACLASS and ATTRIBUTE properties for Windows controls that will be propagated to the new PDML file by following a simple naming convention when you create the identifiers for your Windows resources. These properties will be used to generate source code skeletons for your JavaBeans when you run the conversion.

The naming convention for Windows resource identifiers is:

```
IDCB_<class name>_<attribute>
```

where <class name> is the fully-qualified name of the bean class that you want to designate as the DATACLASS property of the control, and <attribute> is the name of the bean property that you want to designate as the ATTRIBUTE property of the control.

For example, a Windows text field with the resource ID IDCB_com_MyCompany_MyPackage_MyBean_SampleAttribute produces a DATACLASS property of **com.MyCompany.MyPackage.MyBean** and an ATTRIBUTE property of **SampleAttribute**. If you elect to generate JavaBeans when you run the conversion, the Java source file **MyBean.java** is produced, containing the package statement **package com.MyCompany.MyPackage**, and gettor and settor methods for the **SampleAttribute** property.

Displaying your panels at runtime

The Graphical Toolbox provides a redistributable API that your Java programs can use to display user interface panels defined using PDML. The API displays your panels by interpreting the PDML and rendering your user interface using the Java Foundation Classes.

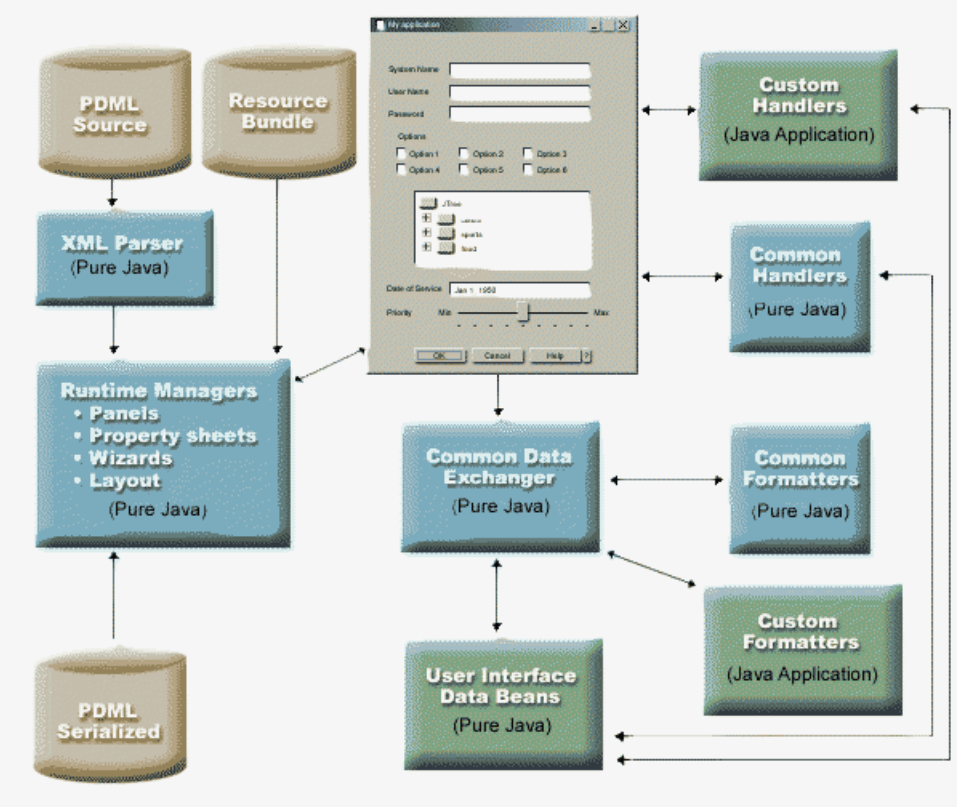
The Graphical Toolbox runtime environment provides the following services:

- Handles all data exchanges between user interface controls and the JavaBeans that you identified in the PDML.
- Performs validation of user data for common integer and character data types, and defines an interface that allows you to implement custom validation. If data is found to be invalid, an error message is displayed to the user.
- Defines standardized processing for Commit, Cancel and Help events, and provides a framework for handling custom events.
- Manages interactions between user interface controls based on state information defined in the PDML. (For example, you may want to disable a group of controls whenever the user selects a particular radio button.)

The package com.ibm.as400.ui.framework.java contains the Graphical Toolbox runtime API.

The elements of the Graphical Toolbox runtime environment are shown in Figure 1. Your Java program is a client of one or more of the objects in the **Runtime Managers** box.

Figure 1: Graphical Toolbox Runtime Environment



Examples

Assume that the panel **MyPanel** is defined in the file **TestPanels.pdml**, and that a properties file **TestPanels.properties** is associated with the panel definition. Both files reside in the directory **com/ourCompany/ourPackage**, which is accessible either from a directory defined in the classpath or from a ZIP or JAR file defined in the classpath.

Note: Read the Code example disclaimer for important legal information.

Example: Creating and displaying a panel

The following code creates and displays the panel:

```
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);
```

Example: Creating a dialog

Once the DataBeans that supply data to the panel have been implemented and the attributes have been identified in the PDML, the following code may be used to construct a fully-functioning dialog:

```
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);
```

Example: Using the dynamic panel manager

A new service has been added to the existing panel manager. The dynamic panel manager dynamically sizes the panel at runtime. Let's look at the **MyPanel** example again, using the dynamic panel manager:

```
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);
```

When you instantiate this panel application you can see the dynamic sizing feature of the panels. Move your cursor to the edge of the GUI's display and, when you see the sizing arrows, you can change the size of the panel.

Long description of Figure 1: Graphical Toolbox runtime environment (rzahh504.gif)

found in IBM Toolbox for Java: Displaying your panels at runtime

This figure illustrates how the elements of the Graphical Toolbox runtime environment interact with application code.

Description

The figure is composed of several boxes of differing shapes, sizes, and colors that are connected to each other by lines terminated by arrowheads at one or both ends.

In order to visualize the figure, it is useful to divide it into three columns and four rows, numbering the areas in sequence from top left to bottom right. fo example, The first row contains areas 1, 2, and 3; the second row contains areas 4, 5, and 6; and so on:

- The image of a dialog box that occupies areas 2 and 5 represents the GUI interface for your Java program. The dialog box features a variety of options, like check boxes, text fields, and so on.
- Two tan cylinders at the top of area 1 are labeled PDML Source and Resource Bundle. These cylinders represent PDML source and Java resource files that reside on a storage medium.
- One tan cylinder in area 10 labeled PDML Serialized represents one or more serialized PDML files that reside on a storage medium.
- Five blue rectangles that surround the bottom portion of the dialog box represent components of the Graphical Toolbox. Starting at the leftmost rectangle and moving counter-clockwise, they are labeled:
  - XML Parser (Pure Java) in area 4, which represents the IBM XML Parser.
  - Runtime Managers (Pure Java) in area 7. Your Java program is a client of one or more of the objects contained in Runtime Managers: Panels, Property sheets, Wizards, and Layout.
  - Common Data Exchanger (Pure Java) in area 8.
  - Common Formatters (Pure Java) in area 9.
  - Common Handlers (Pure Java) in area 6.
- Three green rectangles represent code provided by the application programmer and are labeled:
  - Custom Handlers (Java Application) in area 3
  - Custom Formatters (Java Application) in are 12
  - User Interface Data Beans (Pure Java) in area 11
- Lines connect many of the shapes:
  - A line that has a single arrowhead (on one end) indicates an action. Single arrowhead lines point toward a function or component that uses the object from which the line originates. In the following description, the word ″use″ means that a line with a single arrowhead points toward an object from the component that acts upon it.
  - A line that has a double arrowhead (one at each end) indicates an interaction. These lines connect objects that share a two-way exchange of information. In the following description, the word ″interact″ means that the components are connected by a line with a double arrowhead.

The GUI interface for your Java program (the image of the dialog in areas 2 and 5) interacts with the Runtime Managers for the Graphical Toolbox (the blue rectangle in area 7).

The Runtime Managers, which are pure Java, contain panels, property sheets, wizards, and the GUI layouts. To generate the GUI, the Runtime Managers use a Java resource bundle (one of two tan cylinders in area 1) and PDML data. Runtime Managers can process PDML data in one of two ways:

- Using serialized PDML files (the tan cylinder in area 10)
- Using the IBM iSeries XML Parser (the blue rectangle in area 4), which in turn uses (parses) the PDML source files (one of two tan cylinders in area 1)

Your GUI-enabled Java program operates on data in one of the following ways:

- Having the GUI interface interact with custom handlers (the green rectangle in area 3) and common handlers (the blue rectangle in area 6)
- Having the common data exchanger (the blue rectangle in area 8) use the GUI interface to obtain information

The custom handlers, common handlers, and the common data exchanger all interact with the user interface data beans (the green rectangle in area 11), passing information back and forth. The common data exchanger interacts with common formatters (the blue rectangle in area 9) and custom formatters (the green rectangle in area 12) to convert the data into appropriate formats for the user interface data beans.

Editing help documents generated by GUI Builder

For each PDML project file, the GUI Builder generates a help skeleton and puts it into a single HTML document. Before use, this HTML file is broken up into single topic HTML files for each dialog of the PDML project. This provides the user with granular help for each topic and allows you to manage only a few large help files.

The Help Document is a valid HTML file and can be viewed in any browser and edited using most HTML editors. Tags that define the sections in a Help Document are embedded within comments, so they do not show up in a browser. The comment tags are used to break the Help Document into several sections:

- Header
- Topic section for each dialog
- Topic section for each control that is help-enabled
- Footer

In addition, you can add additional topic sections before the footer to provide additional information or common information. Topic sections have only the html body until they are split, when a header and footer are created. When the Help Document is split up, the processor adds a header and footer to the topic section to make a complete HTML file. The header and footer from the Help Document are used as default header and footer. However, you can override the default header with your own.

Inside the Help Document

The following sections explain the parts of the Help Document:

Header

The end of the header section is shown by the following tag:

```
<!-- HELPDOC:HEADEREND -->
```

If you want to override the default header for all of the individual topics when they are split, use the HEADER keyword and provide the name of an html fragment to include. For example:

```
<!-- HELPDOC:HEADEREND HEADER="defaultheader.html" -->
```

Topic segment

Each topic is surrounded by the following tags:

```
<!-- HELPDOC:SEGMENTBEGIN -->
```

and

```
<!-- HELPDOC:SEGMENTEND -->
```

Immediately following the SEGMENTBEGIN tag is an anchor tag which names the segment. It also provides the file name of the HTML document that is created when the Help Document is split. The name of the segment combines the panel identifier, control identifier, and future file extension (html). For example: ″MY_PANEL.MY_CONTROL.html″ Segments for panels have only the panel identifier and future file extension.

The help generator will place text in the document indicating where you place your help information:

```
<!-- 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 -->
```

You can add additional HTML 2.0 tags as needed after the anchor tag and before the SEGMENTEND tag.

The PDMLSYNCH tag controls how closely a segment is tied to the controls defined in PDML. If PDMLSYCH is ″YES″, the Help Document segment will be removed if the control of the same name is removed in the PDML. PDMLSYNCH=″NO″ indicates the topic must be kept in the Help Document regardless of whether a corresponding control exists in the PDML. This is used, for example, when you create additional topics for depth or a common topic.

The help generated for a panel has links to each control enabled for help on the panel. These links are generated with a local anchor reference, so that you can test them as internal links in a standard browser. When the Help Document is split, the processor removes the ″#″ on these internal links making them external links in the resulting single topic HTML files. Because you may want to have internal links within a topic, the processor only removes any preceding ″#″ when the reference has ″.html″ embedded in it.

If you want to override the default header for any particular topic, use the HEADER keyword and provide the name of an html fragment to include. For example:

```
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" HEADER="specialheader.html" -->
```

Footer The footer in the Help Document begins with the following tag:

```
<!-- HELPDOC:FOOTERBEGIN -->
```

The standard footer is </BODY></HTML>This footer is added to each HTML file.

Adding links

You can add links to any external or internal URL as well as any other segment. However, you must follow some conventions:

- External URLs are used in the standard manner. This includes internal links to external URLs
- Internal links within the same topic are written in the standard way, but must not have ″.html″ as part of the tag name. This is because the Help Document processor assumes that any link with .html will need to be an external link when the topics are separate. Therefore, it removes the preceding ″#″.
- Links to other topic segments must be written with a preceding ″#″ as though they are an internal anchor reference.
- Internal links to other topic segments may also be created. Only the leading ″#″ is removed during processing.

Note:

- At run-time, the PanelManager class looks for help files in a subdirectory with the same name as the PDML file. When the processor splits the Help Document, it creates this subdirectory by default and places the resulting HTML files in it.
- The processor does not make any adjustments for external URL references that are relative links. When you link from an individual topic file, any relative links will be searching from the new subdirectory. Therefore, you will need to place copies of resources such as images where they can be found or use "../" in the path in order to search from the panel directory.

Editing using a visual editor

You can edit your help content in almost any visual HTML editor. Because the HELPDOC tags are comments they may not be obvious in some editors. For convenience, a horizontal rule is added to the help skeleton immediately before the SEGMENTBEGIN tag and immediately after the SEGMENTEND tag. These horizontal rules provide clear visual indication of the entire segment in a visual editor. If you select a segment because you want to move, copy, or delete it, select the surrounding horizontal rules to be sure you have included the SEGMENTBEGIN and SEGMENTEND tags in your selection. These horizontal rules are not copied to the final individual HTML files.

Creating Additional Topics

You can create additional topic segments in the Help Document. It is often easiest to do this by copying another segment. When you copy the segment, you must copy the horizontal rules just before the SEGMENTBEGIN and after the SEGMENTEND tag. This will make future visual editing much easier and help avoid mismatched tags. For best results, use the following tips:

- The name of the anchor must be the name you want for the resulting single file when the Help Document is split. It must end in ″.html″.
- Use the PDMLSYNCH=″NO″ keyword on the SEGMENTBEGIN tag to prevent the segment from being removed if the help skeleton is regenerated.
- Any references to your new topic will be made as an internal link in the Help Document with a preceding ″#″. This ″#″ will be removed in later processing when the segments are split into single files.

Checking Your Links

For most writing, you can check your links by viewing your document in a Web browser and selecting different links. In the single Help Document, the links are still in their internal form.

As you reach completion, or when you want to test with the application you are developing help for, you will need to break the Help Document into single files. You do this with Help Document to HTML Processing.

If you need to regenerate the Help Document after editing, your writing will be preserved. You may want to regenerate the Help Document if you add new controls after generating the original help skeleton. In this case, the help generator checks for an existing Help Document before it creates a new skeleton. If one is found, it preserves any existing segments and then adds the new controls.

Using the Graphical Toolbox in a browser

You can use the Graphical Toolbox to build panels for Java applets that run in a Web browser.

This section describes how to convert the simple panel from the Graphical Toolbox Example to run in a browser. The minimum browser levels supported are Netscape 4.05 and Internet Explorer 4.0. In order to avoid having to deal with the idiosyncrasies of individual browsers, it is recommend that your applets run using Sun's Java Plug-in. Otherwise, you will need to construct signed JAR files for Netscape, and separate signed CAB files for Internet Explorer.

Note: Read the Code example disclaimer for important legal information.

Constructing the applet

The code to display a panel in an applet is nearly identical to the code used in the Java application example, but first, the code must be repackaged in the **init** method of a **JApplet** subclass. Also, some code was added to ensure that the applet panel is sized to the dimensions specified in the panel's PDML definition. Here is the source code for the example applet, **SampleApplet.java**.

```
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);
    }
}
```

The applet's content pane is passed to the Graphical Toolbox as the container to be laid out. In the **start** method, the applet pane is set to its correct size, and then override the **paint** method in order to preserve the panel's size when the browser window is resized.

When running the Graphical Toolbox in a browser, the HTML files for your panel's online help cannot be accessed from a JAR file. They must reside as separate files in the directory where your applet resides. The call to **PanelManager.setHelpPath** identifies this directory to the Graphical Toolbox, so that your help files can be located.

HTML tags

Because it is recommended to use Sun's Java Plug-in to provide the correct level of the Java runtime environment, the HTML for identifying a Graphical Toolbox applet is not as straightforward as preferred. Fortunately, the same HTML template may be reused, with only slight changes, for other applets. The markup is designed to be interpreted in both Netscape Navigator and Internet Explorer, and it generates a prompt for downloading the Java Plug-in from Sun's Web site if it's not already installed on the user's machine. For detailed information on the workings of the Java Plug-in see the Java Plug-in HTML Specification. 

Here is the HTML for the sample applet, in the file **MyGUI.html**:

```
<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>
```

It is important that the version information be set for 1.1.3.

Note: In this example, the XML parser JAR file, **x4j400.jar**, is stored on the Web server. You can use other XML parsers. For more information, see "XML parser and XSLT processor" on page 393. This is required only when you include your PDML file as part of your applet's installation. For performance reasons, you would normally *serialize* your panel definitions so that the Graphical Toolbox does not have to interpret the PDML at runtime. This greatly improves the performance of your user interface by creating compact binary representations of your panels. For more information see the description of files generated by the tools.

Installing and running the applet

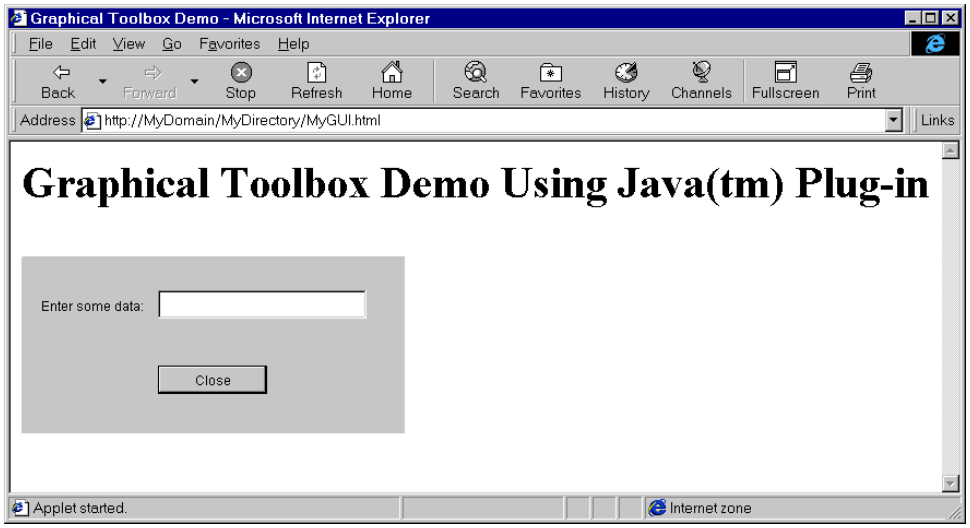
Install the applet on your favorite Web server by performing the following steps:

1. Compile **SampleApplet.java**.
2. Create a JAR file named **MyGUI.jar** to contain the applet binaries. These include the class files produced when you compiled **SampleApplet.java** and **SampleBean.java**, the PDML file **MyGUI.pdml**, and the resource bundle **MyGUI.properties**.
3. Copy your new JAR file to a directory of your choice on your Web server. Copy the HTML files containing your online help into the server directory.
4. Copy the Graphical Toolbox JAR files into the server directory.
5. Finally, copy the HTML file **MyGUI.html** containing the embedded applet into the server directory.

Tip: When testing your applets, ensure that you have removed the Graphical Toolbox jars from the CLASSPATH environment variable on your workstation. Otherwise, you will see error messages saying that the resources for your applet cannot be located on the server.

Now you are ready to run the applet. Point your Web browser to **MyGUI.html** on the server. If you do not already have the Java Plug-in installed, you will be asked if you want to install it. Once the Plug-in is installed and the applet is started, your browser display should look similar to the Figure 1:

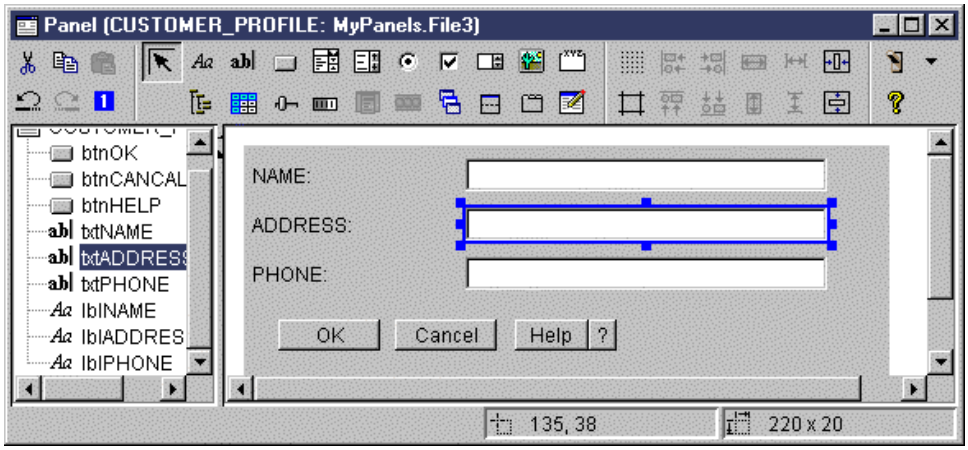
Figure 1: Running the sample applet in a browser



GUI Builder Panel Builder toolbar

Figure 1 shows the GUI Builder Panel Builder window. Following Figure 1 is a list that shows each Panel Builder tool icon and describes its function.

Figure 1: The GUI Builder Panel window



 Click Pointer to move and resize a component on a panel.

 Click Label to insert a static label on a panel.

 Click Text to insert a text box on a panel.

 Click Button to insert a button on a panel.

 Click Combo Box to insert a drop down list box on a panel.

 Click List Box to insert a list box on a panel.

 Click Radio Button to insert a radio button on a panel.

 Click Checkbox to insert a check box on a panel.

 Click Spinner to insert a spinner on a panel.

 Click Image to insert an image on a panel.

 Click Menu Bar to insert a menu bar on a panel.

 Click Group Box to insert a labeled group box on a panel.

 Click Tree to insert an hierarchical tree on a panel.

 Click Table to insert a table on a panel.

 Click Slider to insert an adjustable slider on a panel.

 Click Progress Bar to insert a progress bar on a panel.

 Click Deck Pane to insert a deck pane on a panel. A deck pane contains a stack of panels. The user can select any of the panels, but only the selected panel is fully visible.

 Click Split Pane to insert a split pane on a panel. A split pane is one pane divided into two horizontal or vertical panes.

 Click Tabbed Pane to insert a tabbed pane on a panel. A tabbed pane contains a collection of panels with tabs at the top. The user clicks a tab to display the contents of a panel. The title of the panel is used as the text for a tab.

 Click Custom to insert a custom-defined user interface component on a panel.

 Click Toolbar to insert a toolbar on a panel.

 Click Toggle Grid to enable a grid on a panel.

 Click Align Top to align multiple components on a panel with the top edge of a specific, or primary, component.

 Click Align Bottom to align multiple components on a panel with the bottom edge of a specific, or primary, component.

 Click Equalize Height to equalize the height of multiple components with the height of a specific, or primary, component.

 Click Center Vertically to center a selected component vertically relative to the panel.

 Click Toggle Margins to view the margins of the panel.

 Click Align Left to align multiple components on a panel with the left edge of a specific, or primary, component.

 Click Align Right to align multiple components on a panel with the left edge of a specific, or primary, component.

 Click Equalize Width to equalize the width of multiple components with the width of a specific, or primary, component.

 Click Center Horizontally to center a selected component horizontally relative to the panel.

 Click Cut to cut panel components.

 Click Copy button to copy panel components.

 Click Paste to paste panel components between different panels or files.

 Click Undo to undo the last action.

 Click Redo to redo the last action.

 Click Tab Order to control the selection order of each panel component when the user presses TAB to navigate through the panel.

 Click Preview to display a preview of what a panel will look like.

 Click Help to get more specific information about the Graphical Toolbox.

IBM Toolbox for Java beans

JavaBeans are reuseable software components that are written in Java. The component is a piece of program code that provides a well-defined, functional unit, which can be as small as a label for a button on a window or as large as an entire application.

JavaBeans can be either visual or nonvisual components. Non-visual JavaBeans still have a visual representation, such as an icon or a name, to allow visual manipulation.

Many IBM Toolbox for Java public classes are also JavaBeans. These classes were built to Javasoft JavaBean standards; they function as reuseable components. The properties and methods for an IBM Toolbox for Java bean are the same as the properties and methods of the class.

JavaBeans can be used within an application program or they can be visually manipulated in builder tools, such as the IBM VisualAge for Java product.

Examples

The following examples show how to use JavaBeans in your program and how to create a program from JavaBeans by using a visual bean builder:

"Example: IBM Toolbox for Java bean code" on page 512

"Example: Creating beans with a visual bean builder" on page 513

JDBC

JDBC is an application programming interface (API) included in the Java platform that enables Java programs to connect to a wide range of databases.

The IBM Toolbox for Java JDBC driver allows you to use JDBC API interfaces to issue structured query language (SQL) statements to and process results from databases on the server. You can also use IBM Developer Kit for Java JDBC driver, called the 'native' JDBC driver:

- Use the IBM Toolbox JDBC driver when the Java program is on one system and the database files are on another system, as in a client/server environment
- Use the native JDBC driver when both the Java program and database files are on the same iSeries server

For more information about IBM Toolbox for Java JDBC classes and examples, ongoing improvements, JDBC properties, and unsupported SQL types, see the following pages:

"JDBC classes" on page 61

"Enhancements to JDBC support for Version 5 Release 3" on page 308

"IBM Toolbox for Java JDBC properties" on page 311

"JDBC SQL Types" on page 328

Different versions of JDBC

Different versions of the JDBC API exist, and the IBM Toolbox for Java JDBC driver supports the following versions:

- JDBC 1.2 API (the java.sql package) is included in the Java Platform 1.1 core API and JDK 1.1.
- JDBC 2.1 core API (the java.sql package) is included in both the Java 2 Platform, Standard Edition (J2SE) and the Java 2 Platform Enterprise Edition (J2EE).
- JDBC 2.0 Optional Package API (the javax.sql package) is included in J2EE and is available as a separate download from Sun. These extensions were formerly named the JDBC 2.0 Standard Extension API.
- JDBC 3.0 API (the java.sql and javax.sql packages) is included in J2SE, Version 1.4.

Enhancements to IBM Toolbox for Java JDBC support for V5R4

Several JDBC functions were enhanced for i5/OS Version 5 Release 4.

Enhanced JDBC functions for i5/OS Version 5 Release 4 include:

- "2 MB statement size"
- "128 byte column name support"
- "Database host server trace support"
- "eWLM Correlator support"

For information about enhanced JDBC functions for previous releases, see "Enhancements to JDBC support for Version 5 Release 3" on page 308 and "Enhanced JDBC functions for i5/OS Version 5 Release 2" on page 309.

2 MB statement size

Prior to V5R4, the limit on SQL statement size was 65 535 bytes. This corresponds to 65 535 characters when the statement text is represented using a single-byte CCSID, and 32 767 characters when the statement text is represented using a double-byte CCSID. Some customers, particularly those using applications that automatically generate SQL statements, were affected by this limit.

In V5R4, the iSeries statement size limit has been increased to two megabytes, or 2 097 152 bytes. The IBM Toolbox for Java JDBC driver always sends statement text in two byte Unicode. Therefore, the maximum statement length in characters will be one megabyte or 1 048 576 characters.

128 byte column name support

Starting with V5R4, the database will support column names up to 128 bytes for SQL tables. Prior to V5R4, column names up to 30 bytes were supported. The IBM Toolbox for Java JDBC driver will provide these possibly longer names to its users.

There is one exception where 128 byte column names will not be returned. When local package caching is used and column names exceed 30 characters, the server will return the column names as the system column name.

Database host server trace support

A new option was added to the Toolbox for Java JDBC driver to turn on database host server tracing. To support this feature, option "64" was added to the "server trace" connection property. For more details, see "IBM Toolbox for Java JDBC properties" on page 311.

eWLM Correlator support

The IBM Toolbox for Java will accept an IBM Enterprise Workload Manager (eWLM) correlator and pass it on to the host as a connection attribute correlator for use with the Application Response Measurement (ARM) APIs. This correlator can be sent to the host at any time after a connection is made using the following method in the AS400JDBCConnection class:

setDB2eWLMCorrelator

```
public void setDB2eWLMCorrelator(byte[] bytes)
                          throws SQLException
```

Sets the eWLM Correlator. It is assumed a valid correlator value is used. If the value is null, all ARM/eWLM implementation will be turned off. eWLM correlators require i5/OS V5R3 or later servers. This request is ignored when running to OS/400® V5R2 or earlier servers.

Parameters:

- `bytes`: The eWLM correlator value
- `SQLException`: See the Class SQLException information at the Sun Microsystems, Inc. Web site.

For information about enhanced JDBC functions for previous releases, see "Enhancements to JDBC support for Version 5 Release 3" and "Enhanced JDBC functions for i5/OS Version 5 Release 2" on page 309.

Enhancements to JDBC support for Version 5 Release 3

Several JDBC functions were enhanced for i5/OS Version 5 Release 3.

Enhanced JDBC functions for i5/OS Version 5 Release 3 include:

- UTF-8 and UTF-16 support
- Binary and Varbinary support
- Increased Decimal Precision support
- 2 GB large object support:
- Insensitive cursor support
- Materialized Query Table support

For information about enhanced JDBC functions for previous releases, see V5R2 enhancements to IBM Toolbox for Java JDBC support.

UTF-8 and UTF-16 support

UTF-8 data is stored in a character field with a CCSID of 1208. A UTF-8 character is a variable number of bytes (one, two, three, or four) for a non-combining character, and any number of bytes for a combining character. The length specified for a character field is the maximum number of bytes the field can contain. You can tag the following data types with a UTF-8 1208 CCSID:

- Fixed length character (CHAR)
- Variable length character (VARCHAR)
- Character LOB (CLOB)

UTF-16 data is stored in a graphic field with a CCSID of 1200. A UTF-16 character can be either two or four bytes (that is, Surrogate) in length for a non-combining character and any number of bytes for a combining character. The length specified for a graphic data field is the maximum number of two bytes characters the field can contain. You can tag the following data types with a UTF-16 1200 CCSID:

- Fixed length graphic (GRAPHIC)
- Variable length graphic (VARGRAPHIC)
- Double-byte character LOB (DBCLOB)

Binary and Varbinary support

The BINARY and VARBINARY data types are similar to the CHAR and VARCHAR data types, but contain binary data rather than character data. BINARY fields have a fixed length. VARBINARY fields are of varying length. The BINARY and VARBINARY data types have the following characteristics:

- The coded character set identifier (CCSID) for binary types is 65535
- In assignments and comparisons, binary data types are compatible only with other binary data types (BINARY, VARBINARY, and BLOB)
- The pad character for binary data types is x'00' instead of the blank character
- In situations requiring trailing characters to be stripped to prevent truncation errors, x'00' characters are stripped instead of trailing blanks

- When comparing binary data types, for two fields to be equal both the data and the lengths must be the same. Trailing zeros are not ignored in comparisons
- When comparing binary data types, if two fields have different lengths, the shorter field is considered less than the longer field if the fields are the same up to the length of the shorter field

Increased Decimal Precision support
Decimal precision now supports up to 63 digits. Three properties were added, ″minimum divide scale″, ″maximum precision″, and ″maximum scale″ and six methods added to AS400JDBCDataSource, setMinimumDivideScale(int divideScale), getMinimumDivideScale(), setMaximumPrecision(int precision), getMaximumPrecision(), setMaximumScale(int scale), and getMaximumScale(). Minimum divide scale specifies the minimum scale value for the result of decimal division and is set to any number between 0 and 9. Maximum precision specifies the maximum decimal precision the database uses and is set to either 31 or 63. Maximum scale specifies the maximum scale the database uses and is set to any number between 0 and 63.

2 GB large object (LOB) support
Enhancements for IBM Toolbox for Java JDBC now allow the use of up to 2 GB LOBs support.

Insensitive cursor support
Cursor support now supports insensitive cursors. When using a ResultSet with TYPE_SCROLL_INSENSITIVE, an insensitive cursor is used. The ResultSet does not show changes to the underlying database while it is open.

Materialized Query Table support
Returns ″MATERIALIZED QUERY TABLE″ as the TABLE_TYPE in a call to DatabaseMetaData.getTables().

Enhanced JDBC functions for i5/OS Version 5 Release 2

Several JDBC functions were enhanced for i5/OS Version 5 Release 2.

Enhanced JDBC functions for i5/OS Version 5 Release 2 include:

- Removal of 'FOR UPDATE' restriction
- Change in data truncation
- Get and modify columns and parameters by name
- Retrieve auto-generated keys
- Improved performance when running SQL insert statements in a batch
- Enhanced support for ResultSet.getRow()
- Improved support for using mixed cases in column names
- Specify holdability for Statements, CallableStatements, and PreparedStatements
- Enhanced transaction isolation support

Removal of the 'FOR UPDATE' restriction
You no longer need to specify FOR UPDATE on your SELECT statements in order to guarantee an updatable cursor. When connecting to V5R1 and later versions of i5/OS, IBM Toolbox for Java honors whatever concurrency you pass in when you create statements. The default continues to be a read-only cursor if you do not specify a concurrency.

Data truncation throws exceptions only when truncated character data is written to the database
Data truncation rules for IBM Toolbox for Java now are the same as those for the IBM Developer Kit for Java JDBC driver. For more information, see IBM Toolbox for Java JDBC properties.

Get and modify columns and parameters by name
New methods allow you to get and update information by column name in ResultSet and to get and set information by parameter name in CallableStatement. For example, in ResultSet, where you previously used the following:

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

You can now use:

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

Be aware that accessing parameters by their index results in better performance than accessing them by their name. You can also specify parameter names to set in CallableStatement. Where you might have used the following in CallableStatement:

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

You can now use:

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

To use these new methods, you need JDBC 3.0 or later and the Java 2 Platform, version 1.4 (either the Standard or the Enterprise Edition).

Retrieve auto-generated keys

The getGeneratedKeys() method on AS400JDBCStatement retrieves any auto-generated keys created as a result of executing that Statement object. When the Statement object does not generate any keys, an empty ResultSet object is returned. Currently the server supports returning only one auto-generated key (the key for the last inserted row). The following example shows how you might insert a value into a table then get the auto-generated key:

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

To retrieve auto-generated keys, you need JDBC 3.0 or later, and the Java 2 Platform, version 1.4 (either the Standard or the Enterprise Edition). Retrieving auto-generated keys also requires connecting to a V5R2 or later version of i5/OS.

Improved performance when running SQL insert statements in a batch

Performance of running SQL insert statements in a batch has been improved. Run SQL statements in a batch by using the different addBatch() methods available in AS400JDBCStatement, AS400JDBCPreparedStatement, and AS400JDBCCallableStatement. Enhanced batch support affects only insert requests. For example, using batch support to process several inserts involves only one pass to the server. However, using batch support to process an insert, and update, and a delete sends each request individually.

To use batch support, you need JDBC 2.0 or later and the Java 2 Platform, version 1.2 (either the Standard or the Enterprise Edition).

Enhanced support for ResultSet.getRow()

Previously, the IBM Toolbox for Java JDBC driver was limited in its support for the getRow() method in ResultSet. Specifically, using ResultSet.last(), ResultSet.afterLast(), and ResultSet.absolute() with a negative value made the current row number not available. The previous restrictions are lifted, making this method fully functional.

Using mixed case in column names

IBM Toolbox for Java methods must match either column names provided by the user or column names provided by the application with the names that are on the database table. In either case, when a column name is not enclosed in quotes, IBM Toolbox for Java changes the name to

uppercase characters before matching it against the names on the server. When the column name is enclosed in quotes, it must exactly match the name on the server or IBM Toolbox for Java throws an exception.

Specify holdability in created Statements, CallableStatements, and PreparedStatements
New methods in AS400JDBCConnection allow you to specify the holdability for Statements, CallableStatements, and PreparedStatements that you create. Holdability determines whether cursors are held open or closed when committing the transaction. You can now have a statement that has a different holdability than its connection object. Also, connection objects can have multiple open statement objects, each with a different specified holdability. Calling commit causes each statement to be handled according to the holdability specified for that statement.

Holdability is derived in the following order of precedence:

1. Holdability specified on statement creation by using the Connection class methods createStatement(), prepareCall(), or prepareStatement().
2. Holdability specified by using Connection.setHoldability(int).
3. Holdability specified by the IBM Toolbox for Java JDBC cursor hold property (when methods in 1. or 2. are not used)

To use these methods, you need JDBC 3.0 or later, and the Java 2 Platform, version 1.4 (either the Standard or the Enterprise Edition). Also, servers running a V5R1 or earlier version of i5/OS are able to use only the holdability specified by the JDBC cursor hold property.

Enhanced transaction isolation support
The IBM Toolbox for Java JDBC driver now features support for switching to a transaction isolation level of *NONE after a connection is made. Before V5R2, the IBM Toolbox for Java JDBC driver threw an exception when switching to *NONE after making a connection.

IBM Toolbox for Java JDBC properties

Many properties can be specified when connecting to a server database using JDBC. All properties are optional and can be specified either as part of the URL or in a java.util.Properties object. If a property is set in both the URL and a Properties object, the value in the URL will be used.

Note: The following list does not include DataSource properties.

The following tables list the different connection properties that are recognized by this driver. Some of these properties affect performance and others are server job attributes. The tables organize the properties into the following categories:

General properties

General properties are system attributes that specify the user, password, and whether a prompt is necessary to connect to the server.

General property	Description	Required	Choices	Default
"password"	Specifies the password for connecting to the server. If none is specified, then the user will be prompted, unless the "prompt" property is set to "false", in which case an attempt to connect will fail.	no	server password	(user will be prompted)
"prompt"	Specifies whether the user is prompted if a user name or password is needed to connect to the server. If a connection cannot be made without prompting the user, and this property is set to "false", then an attempt to connect will fail.	no	"true" "false"	"true"
"user"	Specifies the user name for connecting to the server. If none is specified, then the user will be prompted, unless the "prompt" property is set to "false", in which case an attempt to connect will fail.	no	server user	(user will be prompted)

Server properties

Server properties specify attributes that govern transactions, libraries, and databases.

Server property	Description	Required	Choices	Default
"cursor hold"	Specifies whether to hold the cursor across transactions. If this property is set to "true", cursors are not closed when a transaction is committed or rolled back. All resources acquired during the unit of work are held, but locks on specific rows and objects implicitly acquired during the unit of work are released.	no	"true" "false"	"true"
"cursor sensitivity"	Specifies the cursor sensitivity to request from the database. The behavior depends on the resultSetType: • ResultSet.TYPE_FORWARD_ONLY or ResultSet.TYPE_SCROLL_SENSITIVE means that the value of this property controls what cursor sensitivity the Java program requests from the database. • ResultSet.TYPE_SCROLL_INSENSITIVE causes this property to be ignored. This property is ignored when connecting to systems running V5R1 and earlier versions of i5/OS.	no	"" (Use the ResultSet Type to determine the cursor sensitivity) "asensitive" "sensitive" "insensitive"	""

Server property	Description	Required	Choices	Default
″database name″	Specifies the database to use for the connection, including one stored in an independent auxiliary storage pool. This property applies only when connecting to a V5R2 or later version of i5/OS. When you specify a database name, the name must exist in the relational database directory on the server. The following criteria determine which database is accessed: • When this property is used to specify a database, the specified database is used. When the specified database does not exist, the connection fails. • When this property is used to specify *SYSBAS as the database name, the system default database is used. • When this property is omitted, the database name specified in the job description for the user profile is used. When the job description does not specify a database name, the system default database is used.	no	Database name ″*SYSBAS″	The database name specified in the job description for the user profile is used. When the job description does not specify a database name, the system default database is used.

Server property	Description	Required	Choices	Default
″libraries″	Specifies one or more libraries that you want to add to or replace the library list of the server job, and optionally sets the default library (default schema). **Library list** The server uses specified libraries to resolve unqualified stored procedure names, and stored procedures use them to resolve unqualified names. To specify multiple libraries, use commas or spaces to separate individual entries. You can use *LIBL as a placeholder for the current library list of the server job: • When the first entry is *LIBL, the specified libraries are added to the current library list of the server job • When you do not use *LIBL, the specified libraries replace the current library list of the server job For more information about library list properties, see JDBC LibraryList property. **Default schema** The server uses the default schema to resolve unqualified names in SQL statements. For example, in the statement ″SELECT * FROM MYTABLE″, the server looks only in the default schema for MYTABLE. You can specify the default schema on the connection URL. When you do not specify the default schema on the connection URL, the following conditions apply, depending on whether you use SQL Naming or System Naming. • **SQL Naming** When you do not specify the default schema on the connection URL: – The first entry (unless it is *LIBL) becomes the default schema – When the first entry is *LIBL, the second entry becomes the default schema – When you do not set this property or when it contains only *LIBL, the user profile becomes the default schema • **System Naming** When you do not specify the default schema on the connection URL: – No default schema is set, and the server uses the specified libraries to search for unqualified names – When you do not set this property or when it contains only *LIBL, the server uses the current library list of the server job to search for unqualified names	no	List of server libraries, separated by commas or spaces	″*LIBL″
″maximum precision″	Specifies the maximum decimal precision the database might use.	no	″31″ ″63″	″31″
″maximum scale″	Specifies the maximum scale the database might use.	no	″0″-″63″	″31″

Server property	Description	Required	Choices	Default
″minimum divide scale″	Specifies the minimum scale value for the result of decimal division.	no	″0″ ″1″ ″2″ ″3″ ″4″ ″5″ ″6″ ″7″ ″8″ ″9″	″0″
″package ccsid″	Specifies the character encoding to use for the SQL package and any statements sent to the server.	no	″1200″ (UCS-2) ″13488″ (UTF-16)	″13488″
″rollback cursor hold″	Specifies whether to hold the cursor after a rollback. If this property is set to ″true″, cursors are held after a transaction is rolled back.	no	″true″ ″false″	″false″
″transaction isolation″	Specifies the default transaction isolation.	no	″none ″read uncommitted″ ″read committed″ ″repeatable read″ ″serializable″	″read uncommitted″
″translate hex″	Specifies how hexadecimal literals are interpreted.	no	″character″ (Interpret hexadecimal literals as character data) ″binary″ (Interpret hexadecimal literals as binary data)	″character″

Server property	Description	Required	Choices	Default
"true autocommit"	Specifies whether the connection should use true auto commit support. True autocommit means that autocommit is on and is running under a isolation level other than *NONE. By default, the driver handles autocommit by running under the server isolation level of *NONE.	no	"true" (Use true autocommit.) "false" (Do not use true autocommit.)	"false"
"xa loosely coupled support"	Specifies whether lock sharing is allowed for loosely coupled transaction branches. **Note:** This setting is ignored when running to i5/OS V5R3 or earlier.	no	"0" = Locks cannot be shared "1" = Locks can be shared	"0"

Format properties

Format properties specify date and time formats, date and decimal separators, and table naming conventions used within SQL statements.

Format property	Description	Required	Choices	Default
"date format"	Specifies the date format used in date literals within SQL statements.	no	"mdy" "dmy" "ymd" "usa" "iso" "eur" "jis" "julian"	(server job)
"date separator"	Specifies the date separator used in date literals within SQL statements. This property has no effect unless the "date format" property is set to "julian", "mdy", "dmy" or "ymd".	no	"/" (slash) "-" (dash) "." (period) "," (comma) "b" (space)	(server job)
"decimal separator"	Specifies the decimal separator used in numeric literals within SQL statements.	no	"." (period) "," (comma)	(server job)
"naming"	Specifies the naming convention used when referring to tables.	no	"sql" (as in `schema.table`) "system" (as in `schema/table`)	"sql"

Format property	Description	Required	Choices	Default
″time format″	Specifies the time format used in time literals within SQL statements.	no	″hms″ ″usa″ ″iso″ ″eur″ ″jis″	(server job)
″time separator″	Specifies the time separator used in time literals within SQL statements. This property has no effect unless the ″time format″ property is set to ″hms″.	no	″:″ (colon) ″.″ (period) ″,″ (comma) ″b″ (space)	(server job)

Performance properties

Performance properties are attributes that include caching, data conversion, data compression, and prefetching that affect performance.

Performance property	Description	Required	Choices	Default
″big decimal″	Specifies whether an intermediate java.math.BigDecimal object is used for packed and zoned decimal conversions. If this property is set to ″true″, an intermediate java.math.BigDecimal object is used for packed and zoned decimal conversions as described by the JDBC specification. If this property is set to ″false″, no intermediate objects are used for packed and zoned decimal conversions. Instead, such values are converted directly to and from Java double values. Such conversions will be faster but may not follow all conversion and data truncation rules documented by the JDBC specification.	no	″true″ ″false″	″true″
″block criteria″	Specifies the criteria for retrieving data from the server in blocks of records. Specifying a non-zero value for this property will reduce the frequency of communication to the server, and therefore improve performance. Ensure that record blocking is off if the cursor is going to be used for subsequent UPDATEs, or else the row that is updated will not necessarily be the current row.	no	″0″ (no record blocking) ″1″ (block if FOR FETCH ONLY is specified) ″2″ (block unless FOR UPDATE is specified)	″2″

Performance property	Description	Required	Choices	Default
″block size″	Specifies the block size (in kilobytes) to retrieve from the server and cache on the client. This property has no effect unless the ″block criteria″ property is non-zero. Larger block sizes reduce the frequency of communication to the server, and therefore may improve performance.	no	″0″ ″8″ ″16″ ″32″ ″64″ ″128″ ″256″ ″512″	″32″
″data compression″	Specifies whether result set data is compressed. If this property is set to ″true″, then result set data is compressed. If this property is set to ″false″, then result set data is not compressed. Data compression may improve performance when retrieving large result sets.	no	″true″ ″false″	″true″
″extended dynamic″	Specifies whether to use extended dynamic support. Extended dynamic support provides a mechanism for caching dynamic SQL statements on the server. The first time a particular SQL statement is prepared, it is stored in a SQL package on the server. If the package does not exist, it is automatically created. On subsequent prepares of the same SQL statement, the server can skip a significant part of the processing by using information stored in the SQL package. If this is set to ″true″, then a package name must be set using the ″package″ property.	no	″true″ ″false″	″false″
″lazy close″	Specifies whether to delay closing cursors until subsequent requests. This will improve overall performance by reducing the total number of requests.	no	″true″ ″false″	″false″
″lob threshold″	Specifies the maximum LOB (large object) size (in bytes) that can be retrieved as part of a result set. LOBs that are larger than this threshold will be retrieved in pieces using extra communication to the server. Larger LOB thresholds will reduce the frequency of communication to the server, but will download more LOB data, even if it is not used. Smaller LOB thresholds may increase frequency of communication to the server, but will only download LOB data as it is needed.	no	″0″ - ″16777216″	″32768″

Performance property	Description	Required	Choices	Default
″package″	Specifies the base name of the SQL package. Note that only the first seven characters are used to generate the name of the SQL package on the server. This property has no effect unless the ″extended dynamic″ property is set to ″true″. In addition, this property must be set if the ″extended dynamic″ property is set to ″true″.	no	SQL package	″″
″package add″	Specifies whether to add newly prepared statements to the SQL package specified on the ″package″ property. This property has no effect unless the ″extended dynamic″ property is set to ″true″.	no	″true″ ″false″	″true″
″package cache″	Specifies whether to cache a subset of the SQL package information in client memory. Caching SQL packages locally reduces the amount of communication to the server for prepares and describes. This property has no effect unless the ″extended dynamic″ property is set to ″true″.	no	″true″ ″false″	″false″
″package criteria″	Specifies the type of SQL statements to be stored in the SQL package. This can be useful to improve the performance of complex join conditions. This property has no effect unless the ″extended dynamic″ property is set to ″true″.	no	″default″ (only store SQL statements with parameter markers in the package) ″select″ (store all SQL SELECT statements in the package)	″default″
″package error″	Specifies the action to take when SQL package errors occur. When a SQL package error occurs, the driver will optionally throw a SQLException or post a notice to the Connection, based on the value of this property. This property has no effect unless the ″extended dynamic″ property is set to ″true″.	no	″exception″ ″warning″ ″none″	″warning″
″package library″	Specifies the library for the SQL package. This property has no effect unless the ″extended dynamic″ property is set to ″true″.	no	Library for SQL package	″QGPL″
″prefetch″	Specifies whether to prefetch data upon executing a SELECT statement. This will improve performance when accessing the initial rows in the ResultSet.	no	″true″ ″false″	″true″
″qaqqinilib″	Specifies a QAQQINI library name. Used to specify the library that contains the qaqqini file to use. A qaqqini file contains all of the attributes that can potentially impact the performance of the DB2 UDB for iSeries database engine.	no	″QAQQINI library name″	(server default)

Performance property	Description	Required	Choices	Default
″query optimize goal″	Specifies the goal the server should use with optimization of queries. This setting corresponds to the server's QAQQINI option called OPTIMIZATION_GOAL. **Note:** This property is ignored when connecting to systems running to i5/OS V5R3 and earlier.	no	″0″ = Optimize query for first block of data (*FIRSTIO) when extended dynamic packages are used; Optimize query for entire result set (*ALLIO) when packages are not used ″1″ = Optimize query for first block of data (*FIRSTIO) ″2″ = Optimize query for entire result set (*ALLIO)	″0″

Sort properties

Sort properties specify how the server performs stores and performs sorts.

Sort property	Description	Required	Choices	Default
″sort″	Specifies how the server sorts records before sending them to the client.	no	″hex″ (base the sort on hexadecimal values) ″job″ (base the sort on the setting for the server job) ″language″ (base the sort on the language set in the ″sort language″ property) ″table″ (base the sort on the sort sequence table set in the ″sort table″ property)	″job″

Sort property	Description	Required	Choices	Default
″sort language″	Specifies a 3-character language id to use for selection of a sort sequence. This property has no effect unless the ″sort″ property is set to ″language″.	no	Language id	ENU
″sort table″	Specifies the library and file name of a sort sequence table stored on the server. This property has no effect unless the ″sort″ property is set to ″table″.	no	Qualified sort table name	″″
″sort weight″	Specifies how the server treats case while sorting records. This property has no effect unless the ″sort″ property is set to ″language″.	no	″shared″ (uppercase and lowercase characters sort as the same character) ″unique″ (uppercase and lowercase characters sort as different characters)	″shared″

Other properties

Other properties are those properties not easily categorized. These properties determine which JDBC driver is used, and specify options related to level of database access, bidirectional string type, data truncation and so on.

Other property	Description	Required	Choices	Default
″access″	Specifies the level of database access for the connection.	no	″all″ (all SQL statements allowed) ″read call″ (SELECT and CALL statements allowed) ″read only″ (SELECT statements only)	″all″
″behavior override″	Specifies which IBM Toolbox for Java JDBC driver behaviors to override. You can change multiple behaviors in combination by adding the constants and passing that sum on this property. Be sure that your application correctly handles the altered behavior.	no	″″ (do not override any behavior) ″1″ (do not throw an exception but instead return null for the result set if Statement.executeQuery() or PreparedStatement. executeQuery() do not return a result set)	″″

Other property	Description	Required	Choices	Default
″bidi string type″	Specifies the output string type of bidirectional data. See BidiStringType for more information.	no	″″ (use the CCSID to determine bidirectional string type) ″0″ (the default string type for nonbidirectional data (LTR)) ″4″ ″5″ ″6″ ″7″ ″8″ ″9″ ″10″ ″11″	″″
″bidi implicit reordering″	Specifies if bidi implicit LTR-RTL reordering should be used.	no	″true″ ″false″	″true″
″bidi numeric reordering″	Specifies if the numeric ordering round trip feature should be used.	no	″true″ ″false″	″false″
″data truncation″	Specifies whether truncation of **character data** generates attention notices and exceptions. When this property is ″true″, the following apply: • Writing truncated character data to the database throws an exception • Using truncated character data in a query posts an attention notice. When this property is ″false″, writing truncated data to the database or using such data in a query generates no exception or attention notice. The default value is ″true″. This property does not affect **numeric data**. Writing truncated numeric data to the database always throws an error and using truncated numeric data in a query always posts attention notices.	no	″true″ ″false″	″true″
″driver″	Specifies the JDBC driver implementation. The IBM Toolbox for Java JDBC driver can use different JDBC driver implementations based on the environment. If the environment is an iSeries JVM on the same server as the database to which the program is connecting, the native IBM Developer Kit for Java JDBC driver can be used. In any other environment, the IBM Toolbox for Java JDBC driver is used. This property has no effect if the ″secondary URL″ property is set.	no	″toolbox″ (use only the IBM Toolbox for Java JDBC driver) ″native″ (use the IBM Developer Kit for Java JDBC driver if running on the server, otherwise use the IBM Toolbox for Java JDBC driver)	″toolbox″

Other property	Description	Required	Choices	Default
″errors″	Specifies the amount of detail to be returned in the message for errors that occur on the server.	no	″basic″ ″full″	″basic″
″extended metadata″	Specifies whether the driver requests extended metadata from the server. Setting this property to true increases the accuracy of the information returned from the following ResultSetMetaData methods: • getColumnLabel(int) • isReadOnly(int) • isSearchable(int) • isWriteable(int) Additionally, setting this property to true enables support for the ResultSetMetaData.getSchemaName(int) method. Setting this property to true may degrade performance because it requires retrieving more information from the server. Leave the property as the default (false) unless you need more specific information from the listed methods. For example, when this property is off (false), ResultSetMetaData.isSearchable(int) always returns ″true″ because because the driver does not have enough information from the server to make a judgment. Turning on this property (true) forces the driver to get the correct data from the server. You can use extended metadata only when connecting to a server running i5/OS V5R2 or later.	no	″true″ ″false″	″false″
″full open″	Specifies whether the server fully opens a file for each query. By default the server optimizes open requests. This optimization improves performance but may fail if a database monitor is active when a query is run more than once. Set the property to true only when identical queries are issued when monitors are active.	no	″true″ ″false″	″false″
″hold input locators″	Specifies whether input locators should be allocated as type hold locators or not hold locators. If the locators are of type hold, they will not be released when a commit is done.	no	″true″ (type hold) ″false″	″true″

Other property	Description	Required	Choices	Default
″hold statements″	Specifies if statements should remain open until a transaction boundary when autocommit is off and they are associated with a LOB locator. By default, all the resources associated with a statement are released when the statement is closed. Set this property to true only when access to a LOB locator is needed after a statement has been closed.	no	″true″ ″false″	″false″
″key ring name″	Specifies the key ring class name used for SSL connections with the server. This property has no effect unless ″secure″ is set to true and a key ring password is set using the ″key ring password″ property.	no	″key ring name″	″″
″key ring password″	Specifies the password for the key ring class used for SSL communications with the server. This property has no effect unless ″secure″ is set to true and a key ring name is set using the ″key ring name″ property.	no	″key ring password″	″″
″proxy server″	Specifies the host name and port of the middle-tier machine where the proxy server is running. The format for this is *hostname*[:*port*], where the port is optional. If this is not set, then the hostname and port are retrieved from the *com.ibm.as400.access.AS400.proxyServer* property. The default port is *3470* (if the connection uses SSL, the default port is *3471*). The ProxyServer must be running on the middle-tier machine. The name of the middle-tier machine is ignored in a two-tier environment.	no	Proxy server host name and port	(value of the proxyServer property, or none if not set)
″remarks″	Specifies the source of the text for REMARKS columns in ResultSets returned by DatabaseMetaData methods.	no	″sql″ (SQL object comment) ″system″ (i5/OS object description)	″system″
″secondary URL″	Specifies the URL to be used for a connection on the middle-tier's DriverManager in a multiple tier environment, if it is different than already specified. This property allows you to use this driver to connect to databases other than the iSeries server. Use a backslash as an escape character before backslashes and semicolons in the URL.	no	JDBC URL	(current JDBC URL)

Other property	Description	Required	Choices	Default
″secure″	Specifies whether a Secure Sockets Layer (SSL) connection is used to communicate with the server. SSL connections are only available when connecting to servers at V4R4 or later.	no	″true″ (encrypt all client/server communication) ″false″ (encrypt only the password)	″false″
″server trace″	Specifies the level of tracing of the JDBC server job. When tracing is enabled, tracing starts when the client connects to the server and ends when the connection is disconnected. You must start tracing before connecting to the server, because the client enables server tracing only at connect time.	no	″0″ (trace is not active) ″2″ (start the database monitor on the JDBC server job) ″4″ (start debug on the JDBC server job) ″8″ (save the job log when the JDBC server job ends) ″16″ (start job trace on the JDBC server job) ″32″ (save SQL information) ″64″ (supports the activation of database host server tracing) Multiple types of trace can be started by adding these values together. For example, ″6″ starts the database monitor and starts debug.	″0″
″thread used″	Specifies whether threads are used in communication with the host servers.	no	″true″ ″false″	″true″

Other property	Description	Required	Choices	Default
″toolbox trace″	Specifies what category of an IBM Toolbox for Java trace to log. Trace messages are useful for debugging programs that call JDBC. However, there is a performance penalty associated with logging trace messages, so this property is only set for debugging. Trace messages are logged to System.out.	no	″″ ″none″ ″datastream″ (log data flow between the local host and the remote system) ″diagnostic″ (log object state information) ″error″ (log errors that cause an exception) ″information″ (used to track the flow of control through the code) ″warning″ (log errors that are recoverable) ″conversion″ (log character set conversions between Unicode and native code pages) ″proxy″ (log data flow between the client and the proxy server) ″pcml″ (used to determine how PCML interprets the data that is sent to and from the server) ″jdbc″ (log jdbc information) ″all″ (log all categories) ″thread″ (log thread information)	″″
″trace″	Specifies whether trace messages are logged. Trace messages are useful for debugging programs that call JDBC. However, there is a performance penalty associated with logging trace messages, so this property only set to ″true″ for debugging. Trace messages are logged to System.out.	no	″true″ ″false″	″false″
″translate binary″	Specifies whether binary data is translated. If this property is set to ″true″, then BINARY and VARBINARY fields are treated as CHAR and VARCHAR fields.	no	″true″ ″false″	″false″

JDBC Librarylist property

The JDBC LibraryList property specifies one or more libraries that you want to add to or replace the library list of the server job, and optionally sets the default library (default schema).

The examples in the following table make these assumptions:

- A library called MYLIBDAW contains MYFILE_DAW
- You are running this SQL statment:

```
"SELECT * FROM MYFILE_DAW"
```

Scenario	SQL Naming	System Naming
Basic Rules	Only one library is searched. • If a library is specified on the URL, it is used. It becomes the default library. • If no library is specified on the URL, the first library in the 'libraries' property is used. It becomes the default library. • If no library on URL and if no libraries property is specified, the library with the same name as the signed-on user profile is used. The job's library list is updated with the libraries in the libraries property. This may affect the behavior of some triggers and stored procedures. It does not affect unqualified names in statements.	The job's library list is updated with the libraries on the libraries property. If a default library is specified on the URL then that becomes the default library.
1. No library specified anyplace.	Default schema is the user profile name.	No default schema. Job's library list is searched.
2. Default library specified on URL.	Default schema is the specified library.	Default schema is the specified library. Library list is not searched to resolve unqualified name in SQL statements.
3. Default library specified via property.	Default schema is the specified library.	No default schema. All libraries on list searched.
4. Default library specified on URL and property.	Default schema is the library specified on the URL. Library list is ignored.	Default schema is the library specified on the URL. Library list is not searched to resolve unqualified name in SQL statements.
5. Library property specified, lib name is bad	Default schema is the specified library	No default schema. The Library list can not be changed because one of the libraries in the list is not found so the job's library list is used.
6. No library on URL, library property specified, file found in second library in list	Default schema is the first library in the list, rest of libraries ignored.	If all libraries exist , then no default schema, all libraries on list searched, list replaces job's library list. If one of the libraries on the list does not exist, the job's library list is not changed.
7. Library property specified, list starts with a comma	Default schema is user profile	No default schema, all libraries on list searched, list replaces job's library list.
8. Library property specified, list starts with a *LIBL	Default schema is user profile	No default schema, all libraries on list searched, libraries specified added to end of list
9. Library property specified, list ends with a *LIBL	Default schema is first lib on list, rest of list ignored	No default schema, all libraries on list searched, libraries specified added to beginning of job's library list
10. URL library invalid	No default schema, user profile used	No default schema, job's library list is used

Note: When a default schema is specified on the URL and the libraries property is not used, the default schema is appended before the current library list

JDBC SQL Types

Not all of the SQL types described by the JDBC specification are supported by DB2 for i5/OS.

Unsupported SQL Types

In the cases where a SQL type is not supported, the JDBC driver substitutes a similar SQL type.

The following table lists the SQL types that are not supported and the SQL type that JDBC driver substitutes for each.

Unsupported SQL type	Substituted SQL type
BIT	SMALLINT
TINYINT	SMALLINT
BIGINT (on i5/OS Version 4 Release 4 and previous)	INTEGER
LONGVARCHAR	VARCHAR
LONGVARBINARY	VARBINARY

Note: BIGINT is supported on i5/OS V4R5 and later.

Proxy Support

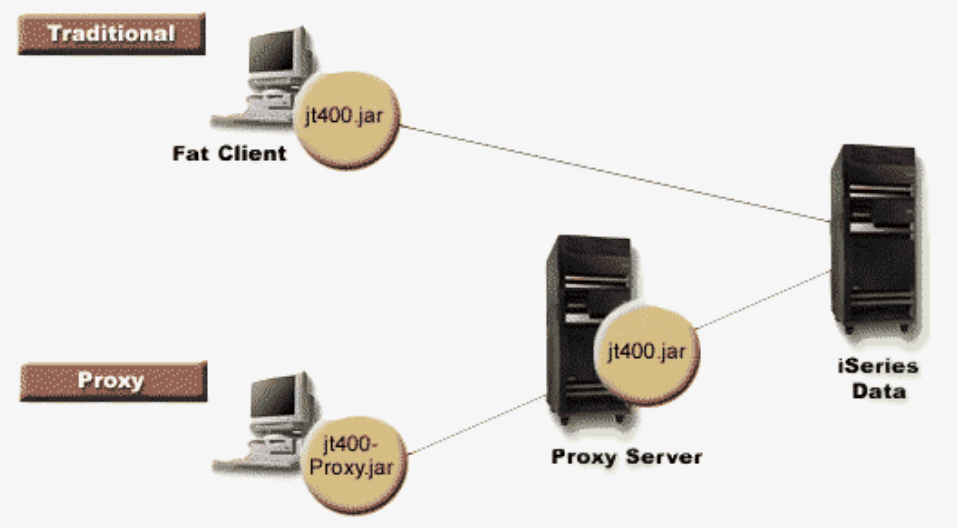
IBM Toolbox for Java includes proxy support for some classes. Proxy support is the processing that IBM Toolbox for Java needs to carry out a task on a Java virtual machine (JVM) when the application is on a different JVM.

Proxy support includes using the Secure Sockets Layer (SSL) protocol to encrypt data.

The proxy classes reside in jt400Proxy.jar, which ships with the rest of the IBM Toolbox for Java. The proxy classes, like the other classes in the IBM Toolbox for Java, comprise a set of platform independent Java classes that can run on any computer with a Java virtual machine. The proxy classes dispatch all method calls to a server application, or proxy server. The full IBM Toolbox for Java classes are on the proxy server. When a client uses a proxy class, the request is transferred to the proxy server which creates and administers the real IBM Toolbox for Java objects.

Figure 1 shows how the standard and proxy client connect to the server. The proxy server can be the iSeries that contains the data.

Figure 1: How a standard client and a proxy client connect to a server



An application that uses proxy support performs more slowly than if it uses standard IBM Toolbox for Java classes due to the extra communication needed to support the smaller proxy classes. Applications that make fewer method calls have less performance degradation.

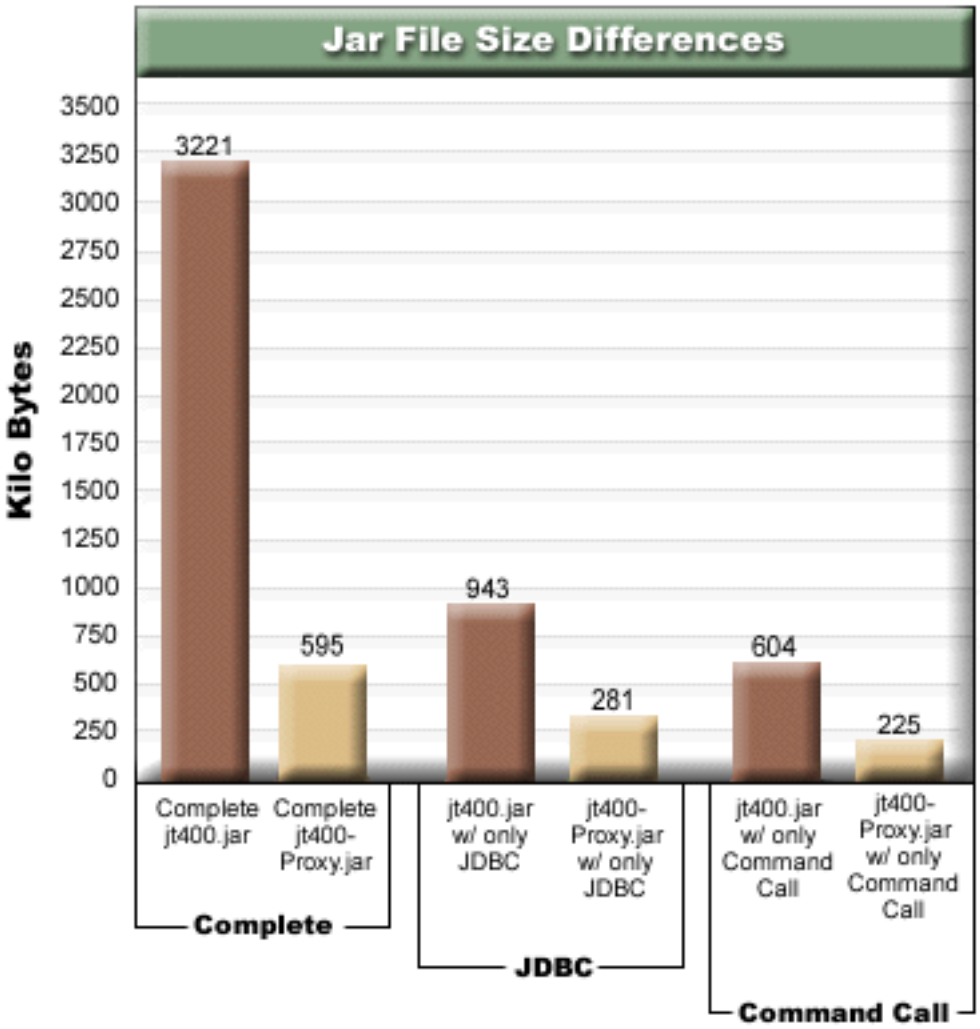
Before proxy support, the classes containing the public interface, all the classes needed to process a request, and the application itself ran on the same JVM. When using proxy support, the public interface must be with the application, but classes for processing requests can run on a different JVM. Proxy support does not change the public interface. The same program can run with either the proxy version of IBM Toolbox for Java or the standard version.

Using the jt400Proxy.jar file

The goal of the multiple-tier, proxy scenario is to make the public interface jar file as small as possible, so that downloading it from an applet takes less time. When you use the proxy classes, you don't need to install the entire IBM Toolbox for Java on the client. Instead, use AS400JarMaker on the jt400Proxy.jar file to include only the required components, which makes the jar file as small as possible.

Figure 2 compares the size of the proxy jar files with the standard jar files:

Figure 2: Size comparison of proxy jar files and standard jar files



An additional benefit is that proxy support requires you to have fewer ports open through a firewall. With standard IBM Toolbox for Java, you must have multiple ports open. This is because each IBM Toolbox for Java service uses a different port to communicate with the server. For example, Command call uses a different port than JDBC, which uses a different port than print, and so on. You must allow each of these ports through the firewall. However, when using proxy support, all the data flows through the same port.

Standard proxy and HTTP tunneling

Two options are available for running via a proxy: standard proxy and HTTP tunneling:

- Standard proxy is where the proxy client and proxy server communicate by using a socket over a port. The default port is 3470. Change the default port by using the setPort() method on the ProxyServer class, or by using the `-port` option when starting the proxy server. For example:

  ```
  java com.ibm.as400.access.ProxyServer -port 1234
  ```

- HTTP tunneling is where the proxy client and proxy server communicate by way of the HTTP server. The IBM Toolbox for Java provides a servlet that handles the proxy request. The proxy client calls the servlet by way of the HTTP server. The advantage of tunneling is that you are not required to open an additional port through the firewalls, because communication is by way of the HTTP port. The disadvantage of tunneling is that it is slower than standard proxy.

IBM Toolbox for Java uses the proxy server name to determine if standard proxy or tunneling proxy is being used:

- For standard proxy, just use the server name. For example:

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

- For tunneling, use a URL to force the proxy client to use tunneling. For example:

  ```
  com.ibm.as400.access.AS400.proxyServer=http://myServer
  ```

When running standard proxy, a socket connection exists between the client and server. If that connection fails, the server cleans up resources associated with that client.

When using HTTP tunneling, using the HTTP protocol makes proxy connectionless. That is, a new connection is made for each data flow. Because the protocol is connectionless, the server does not know if the client application is no longer active. Consequently, the server does not know when to clean up resources. The tunneling server solves this problem by using a thread to clean up resources at a predetermined interval (which is based on a timeout value).

At the end of the predetermined interval, the thread runs and cleans up resources that have not been used lately. Two system properties govern the thread:

- com.ibm.as400.access.TunnelProxyServer. clientCleanupInterval is how often, in seconds, the cleanup thread runs. The default is every two hours.
- com.ibm.as400.access.TunnelProxyServer. clientLifetime is how long, in seconds, a resource can be idle before it is cleaned up. The default is 30 minutes.

Using proxy server

To use the proxy server implementation of the IBM Toolbox for Java classes, complete the following steps:

1. Run AS400ToolboxJarMaker on jt400Proxy.jar to discard classes that you do not need. This step is optional but recommended.
2. Deliver jt400Proxy.jar to the client. For Java applets, you may be able to download the jar file from the HTML server.
3. Determine what server you will use for the proxy server.
   - For Java applications, the proxy server can be any computer.
   - For Java applets, the proxy server must be running on the same computer as the HTTP server.
4. Ensure that you have put jt400.jar in the CLASSPATH on the server.
5. Start the proxy server or use the proxy servlet:
   - For standard proxy, start the proxy server by using the following command:

     ```
     java com.ibm.as400.access.ProxyServer
     ```

   - For tunneling proxy, configure your HTTP server to use the proxy servlet. The servlet class name is com.ibm.as400.access.TunnelProxyServer and it is contained in jt400.jar.
6. On the client, set a system property to identify the proxy server. IBM Toolbox for Java uses this system property to determine if standard proxy or tunneling proxy is being used.
   - For standard proxy, the property value is the name of the machine that runs the proxy server. For example:

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

   - For tunneling proxy, use a URL to force the proxy client to use tunneling. For example:

     ```
     com.ibm.as400.access.AS400.proxyServer=http://myServer
     ```

7. Run the client program.

When you want to work with both the proxy classes and classes not in jt400Proxy.jar, you can refer to jt400.jar instead of jt400Proxy.jar. jt400Proxy.jar is a subset of the jt400.jar and, therefore, all of the proxy classes are contained in the jt400.jar file.

Using SSL

When using proxy, three options are available for encrypting data as it flows from the proxy client to the target iSeries server. SSL algorithms are used to encrypt data.

1. The data flows between the proxy client and proxy server can be encrypted.
2. The data flows between the proxy server and target iSeries server can be encrypted.
3. Both one and two. The data flow between proxy client and proxy server, and the flow between the proxy server and the target iSeries can be encrypted.

See Secure Sockets Layer for more information.

Examples: Using proxy servers

The following are three specific examples for using a proxy server with the steps listed above.

- Running a Java application using proxy support
- Running a Java applet using proxy support
- Running a Java application using tunneling proxy support.

Classes enabled to work with proxy server

Some IBM Toolbox for Java classes are enabled to work with the proxy server application. These include the following:

- JDBC
- Record-level access
- Integrated file system
- Print
- Data Queues
- Command Call
- Program Call
- Service Program Call
- User space
- Data area
- AS400 class
- SecureAS400 class

Other classes are not supported at this time by jt400Proxy. Also, integrated file system permissions are not functional using only the proxy jar file. However, you can use the JarMaker class to include these classes from the jt400.jar file.

Long description of Figure 1: How a standard client and a proxy client connect to a server (rzahh505.gif)

found in IBM Toolbox for Java: Proxy support

This figure illustrates:

- How a standard client and proxy client connect to a server
- The required IBM Toolbox for Java jar files

Description

The figure is composed of the following:

- An image of a personal computer on the upper left that represents a traditional (standard) client. This image includes a circle that contains the name of the required IBM Toolbox for Java jar file (jt400Proxy.jar).
- An image of a personal computer on the lower left that represents a proxy client. This image includes a circle that contains the name of the required IBM Toolbox for Java jar file.
- An image of an iSeries server in the lower center that represents a proxy server. This image includes a circle that contains the name of the required IBM Toolbox for Java jar file.
- An image on an iSeries server on the right that represents the iSeries server. The iSeries server does not require a jar file and so does not include a circle.
- Lines that connect the images together.

The traditional (standard) client bears the label of ″Fat Client″ and uses the jt400.jar file. The standard client connects directly to the iSeries server.

The proxy client uses the jt400Proxy.jar file. The proxy client connects to the proxy server, which uses the jt400.jar file. The proxy server connects to the iSeries server.

Long description of Figure 1: Size comparison of proxy jar files and standard jar files (rzahh502.gif)

found in IBM Toolbox for Java: Proxy support

This figure is a bar chart that compares the sizes of standard and proxy jar files, after using AS400JarMaker to reduce the sizes of the jar files.

Description

The chart makes three comparisons of the jt400.jar file to the jt400Proxy.jar file:

- The complete jar files
- The jar files when they contain only the JDBC component
- The jar files when they contain only the Command Call component

The bar chart is composed of a horizontal axis (x-axis) and a vertical axis (y-axis):

- The horizontal axis (or the x-axis) of the chart is composed of three groups of bars that represent the jar files.
- The vertical axis (or the y-axis) of the chart represents the size of the jar files as measured in kilobytes.

The groups are labeled Complete, JDBC, and Command Call. Each group contains a bar for the jt400.jar file and a bar for the jt400Proxy.jar file. Descriptions for each group follow:

- Complete: The jar file sizes are 3,221 kilobytes for the complete jt400.jar and 595 kilobytes for the complete jt400Proxy.jar.
- JDBC: The jar file sizes are 943 kilobytes for the jt400.jar and 281 kilobytes for the jt400Proxy.jar, each including only the JDBC component.
- Command Call: The jar file sizes are 604 kilobytes for the jt400.jar and 225 kilobytes for the jt400Proxy.jar, each including only the Command Call component.

Example: Running a Java application using Proxy Support

The following example shows you the steps to run a Java application using proxy support.

1. Choose a machine to act as the proxy server. The Java environment and CLASSPATH on the proxy server machine includes the `jt400.jar` file. This machine must be able to connect to the iSeries server.
2. Start the proxy server on this machine by typing: `java com.ibm.as400.access.ProxyServer -verbose` Specifying verbose allows you to monitor when the client connects and disconnects.
3. Choose a machine to act as the client. The Java environment and CLASSPATH on the client machine includes the `jt400Proxy.jar` file and your application classes. This machine must be able to connect to the proxy server but does not need a connection to the iSeries server.
4. Set the value of the `com.ibm.as400.access.AS400.proxyServer` system property to be the name of your proxy server, and run the application. An easy way to do this is by using the -D option on most Java Virtual Machine invocations: `java -Dcom.ibm.as400.access.AS400.proxyServer=psMachineName YourApplication`
5. As your application runs, you see (if you set verbose in step 2) the application make at least one connection to the proxy server.

Example: Running a Java applet using proxy support

The following example shows you the steps to run a Java applet using proxy support.

1. Choose a machine to act as the proxy server. Applets can initiate network connections only to the machine from which they were originally downloaded; therefore, it works best to run the proxy server on the same machine as the HTTP server. The Java environment and CLASSPATH on the proxy server machine includes the jt400.jar file.
2. Start the proxy server on this machine by typing: `java com.ibm.as400.access.ProxyServer -verbose` Specifying verbose will allow you to monitor when the client connects and disconnects.
3. Applet code needs to be downloaded before it runs so it is best to reduce the size of the code as much as possible. The AS400ToolboxJarMaker can reduce the jt400Proxy.jar significantly by including only the code for the components that your applet uses. For instance, if an applet uses only JDBC, reduce the jt400Proxy.jar file to include the minimal amount of code by running the following command:

   ```
   java utilities.AS400ToolboxJarMaker -source jt400Proxy.jar -destination jt400ProxySmall.jar
                                       -component JDBC
   ```

4. The applet must set the value of the com.ibm.as400.access.AS400.proxyServer system property to be the name of your proxy server. A convenient way to do this for applets is using a compiled Properties class (Example). Compile this class and place the generated Properties.class file in the com/ibm/as400/access directory (the same path your html file is coming from). For example, if the html file is /mystuff/HelloWorld.html, then Properties.class is in /mystuff/com/ibm/as400/access.
5. Put the jt400ProxySmall.jar in the same directory as the html file (/mystuff/ in step 4).
6. Refer to the applet like this in your HTML file:

   ```
   <APPLET archive="jt400Proxy.jar, Properties.class" code="YourApplet.class"
           width=300 height=100> </APPLET>
   ```

Example: Running a Java application using Tunneling Proxy Support

The following example shows you the steps to run a Java application using tunneling proxy support.

1. Choose the HTTP server that you want to run the proxy server, then configure it to run servlet com.ibm.as400.access.TunnelProxyServer (in jt400.jar). **Note:** Ensure that the HTTP server has a connection to the iSeries server that contains the data or resource that the application uses because the servlet connects to that iSeries to carry out requests.
2. Choose a machine to act as the client and ensure that the CLASSPATH on the client machine includes the jt400Proxy.jar file and your application classes. The client must be able to connect to the HTTP server but does not need a connection to the iSeries server.
3. Set the value of the com.ibm.as400.access.AS400.proxyServer property to be the name of your HTTP server in URL format.

4. Run the application, setting the value of the com.ibm.as400.access.AS400.proxyServer property to be the name of your HTTP server in URL format.. An easy way to do this is by using the -D option found on most JVMs:

```
java -Dcom.ibm.as400.access.AS400.proxyServer=http://psMachineName YourApplication
```

Note: The proxy client code creates the correct servlet URL by concatenating "servlet" and the servlet name to the server name. In this example, it converts http://psMachineName to http://psMachineName/servlet/TunnelProxyServer

Secure Sockets Layer and Java Secure Socket Extension

IBM Toolbox for Java supports using Java Secure Socket Extension (JSSE) for Java Secure Sockets Layer (SSL) connections. JSSE is available as an optional package to the Java 2 Platform, Standard Edition (J2SE), versions 1.2 and 1.3. JSSE is integrated into J2SE, version 1.4 and subsequent versions.

For more information about JSSE, see the Sun JSSE Web site .

JSSE provides the ability to perform server authentication, enable secure communications, and encrypt data. Using JSSE, you can provide for secure data exchange between clients and servers that run any application protocol (for example, HTTP and FTP) over TCP/IP.

If you have previously used sslight, you should migrate to JSSE. As of V5R4, JSSE is the only package supported, and sslight is no longer shipped.

IBM Toolbox for Java 2 Micro Edition

The IBM Toolbox for Java 2 Micro Edition package (com.ibm.as400.micro) enables you to write Java programs that allow a variety of Tier0 wireless devices, like personal digital assistants (PDAs) and cell phones, to directly access iSeries data and resources.

Downloading and setting up ToolboxME for iSeries

You must separately download ToolboxME for iSeries (jt400Micro.jar), which is contained in JTOpen.

You can download ToolboxME for iSeries from the IBM Toolbox for Java/JTOpen Web site that also offers additional information about setting up ToolboxME for iSeries.

How you set up ToolboxME for iSeries is different for the Tier0 device, the development workstation, and the server:

- Build an application for your wireless device (using jt400Micro.jar) and install the application as documented by the device manufacturer.
- Make sure that the iSeries Host Servers are started on the server that contains the target data.
- Make sure that the system that you want to run the MEServer has access to jt400.jar.

For more information, see the following pages:

"Installing IBM Toolbox for Java on your workstation" on page 10

"Installing IBM Toolbox for Java on an iSeries server" on page 9

Concepts important for using ToolboxME for iSeries

Before you begin developing ToolboxME for iSeries Java applications, you need to understand the following concepts and standards that govern such development.

Java 2 Platform, Micro Edition (J2ME)

The J2ME is the implementation of the Java 2 standard that provides Java runtime environments for Tier0 wireless devices, like personal digital assistants (PDAs) and cell phones. IBM Toolbox for Java 2 Micro Edition adheres to this standard.

Tier0 devices

Wireless devices, such as PDAs and cell phones, that use wireless technology to connect to computers and networks are referred to as Tier0 devices. This name is based on the common 3-tier application model. The 3-tier model describes a distributed program that is organized into three major parts, each of which resides on a different computer or network:

- The third tier is the database and related programs that reside on a server, often a different server than the second tier. This tier provides the information and the access to that information that the other tiers use to perform work.
- The second tier is the business logic, which typically resides on a different computer, typically a server, shared on a network.
- The first tier is generally the part of the application that resides on a workstation, including the user interface.

Tier0 devices are often small, portable, resource-constrained devices, like PDAs and cell phones. Tier0 devices substitute for or complement the functionality of devices on the first tier.

Connected Limited Device Configuration (CLDC)

A configuration defines a minimal set of APIs and the necessary capabilities of a Java virtual machine to provide the functions expected for a large set of devices. The CLDC targets the broad set of resource-constrained devices that include Tier0 devices.

For more information, see CLDC  .

Mobile Information Device Profile (MIDP)

A profile represents a set of APIs built on an existing configuration that target a specific type of device or operating system. The MIDP, built on the CLDC, provides a standard runtime environment that enables you to dynamically deploy applications and services to Tier0 devices.

For more information, see Mobile Information Device Profile (MIDP)  .

Java virtual machine for wireless devices

In order to run Java application, your Tier0 device requires a Java virtual machine that is specially designed for the limited resources of a wireless device. Some of the possible JVMs that you can use include the following:

- IBM J9 virtual machine, part of the IBM WebSphere® Micro Environment 
- Sun K Virtual Machine (KVM), part of CLDC 
- MIDP 

Related information

You can use any one of a number of development tools created to help you build wireless Java applications. For a brief list of such tools, see Related information for IBM Toolbox for Java.

To learn more about and to download wireless device simulators and emulators, consult the Web site for the device or operating system on which you want your application to run.

ToolboxME for iSeries classes

The com.ibm.as400.micro package provides the classes necessary to write applications that enable your Tier0 devices to access iSeries server data and resources.

com.ibm.as400.micro package

"Tier0 devices" on page 336

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component.

ToolboxME for iSeries provides the following classes:

- "MEServer class" mediates requests from the Tier0 device to the host server
- Several classes provide a subset of functions from the IBM Toolbox for Java access package
  - "AS400 class" on page 338 signs on to an iSeries server
  - "CommandCall class" on page 338 calls an iSeries command
  - "DataQueue class" on page 339 reads from and writes to an iSeries server data queue
  - "ProgramCall class" on page 339 calls an iSeries server program and accesses the data that is returned after the program runs
- The "JdbcMe classes" on page 341 provide JDBC support by including the smallest useful set of methods and data from the java.sql package

MEServer class

Use the MEServer class to fulfill requests from your Tier0 client application that uses the ToolboxME for iSeries jar file. The MEServer creates IBM Toolbox for Java objects and invokes methods on them on behalf of the client application.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see Downloading and setting up ToolboxME for iSeries.

Use the following command to start an MEServer:

```
java com.ibm.as400.micro.MEServer [options]
```

where [options] is one or more of the following:

-pcml pcml_doc1 [;pcml_doc2;...]
Specifies the PCML document to preload and parse. You can abbreviate this option by using `-pc`.

For important information about using this option, see the MEServer javadoc.

-port port
Specifies the port to use for accepting connections from clients. The default port is 3470. You can abbreviate this option by using `-po`.

-verbose [true|false]
Specifies whether to print status and connection information to System.out. You can abbreviate this option by using `-v`.

-help Prints usage information to System.out. You can abbreviate this option by using `-h` or `-?`. The default is not to print usage information.

MEServer will not start if another server is already active on the specified port.

AS400 class

The AS400 class in the micro package (com.ibm.as400.micro.AS400) provides a modified subset of the functions available in the AS400 class in the access package (com.ibm.as400.access.AS400). Use the ToolboxMe for iSeries AS400 class to sign on an iSeries server from a Tier0 device.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see Downloading and setting up ToolboxME for iSeries.

The AS400 class provides the following functions:

- Connect to the MEServer
- Disconnect from the MEServer

The connection to the MEServer is made implicitly. For example, after you create an AS400 object, you can use the run() method in CommandCall to automatically perform connect(). In other words, you do not explicitly call the connect() method unless you want to control when the connection is established.

Example: Using the AS400 class

The following example shows how to use the AS400 class to sign on to on an iSeries server:

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    system.connect();
}
catch (Exception e)
{
    // Handle the exception
}
// Done with the system object.
system.disconnect();
```

CommandCall class

The CommandCall class in the micro package (com.ibm.as400.micro.CommandCall) provides a modified subset of the functions available in the CommandCall class in the access package (com.ibm.as400.access.CommandCall). Use the CommandCall class to call an iSeries command from a Tier0 device.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component.

The CommandCall run() method requires a String (the command you want to run) and returns any messages resulting from running the command as a String. If the command completes but does not generate any messages, the run() method returns an empty String array.

Example: Using CommandCall

The following example demonstrates how you can use CommandCall:

```
// Work with commands.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    // Run the command "CRTLIB FRED."
    String[] messages = CommandCall.run(system, "CRTLIB FRED");
    if (messages != null)
    {
        // Note that there was an error.
        System.out.println("Command failed:");
```

```
            for (int i = 0; i < messages.length; ++i)
            {
                System.out.println(messages[i]);
            }
        }
        else
        {
            System.out.println("Command succeeded!");
        }
    }
    catch (Exception e)
    {
        // Handle the exception
    }
    // Done with the system object.
    system.disconnect();
```

DataQueue class

The DataQueue class in the micro package (com.ibm.as400.micro.DataQueue) provides a modified subset of the functions available in the DataQueue class in the access package (com.ibm.as400.access.DataQueue). Use the DataQueue class to have your Tier0 device read from or write to a data queue on the iSeries server.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component.

The DataQueue class includes the following methods:

- Read or write an entry as a String
- Read or write an entry as an array of bytes

To read or write entries, you need to supply the name of the iSeries server where the data queue resides and the fully qualified integrated file system path name of the data queue. When no entries are available, reading an entry returns a null value.

Example: Using DataQueue to read from and write to a data queue

The following example demonstrates how to use the DataQueue class to read entries from and write entries to a data queue on an iSeries server:

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    // Write to the Data Queue.
    DataQueue.write(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ", "some text");

    // Read from the Data Queue.
    String txt = DataQueue.read(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");
}
catch (Exception e)
{
    // Handle the exception
}
// Done with the system object.
system.disconnect();
```

ProgramCall class

The ProgramCall class in the micro package (com.ibm.as400.micro.ProgramCall) provides a modified subset of the functions available in the ProgramCall class in the access package (com.ibm.as400.access.ProgramCall). Use the ProgramCall class to enable a Tier0 device to call an iSeries program and access the data that is returned after the program runs.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see "ToolboxME for iSeries requirements" on page 8.

To use the ProgramCall.run() method, you must provide the following parameters:

- The server on which you want to run the program
- The name of the "Program Call Markup Language" on page 362 document
- The name of the program that you want to run
- The hashtable that contains the name of one or more program parameters that you want to set and the associated values
- The string array that contains the name of any parameters to be returned after the program executes

ProgramCall uses PCML to describe the input and output parameters for the program. The PCML file must be on the same machine as the MEServer, and you must have an entry for the directory that contains the PCML file in the CLASSPATH of that machine.

You must register each PCML document with the MEServer. Registering a PCML document is telling the MEServer which PCML-defined program you want to run. Register the PCML document either during runtime or when you start the MEServer.

For more information about the hashtable that contains program parameters or how to register a PCML document, see the ToolboxME for iSeries ProgramCall javadoc. For more information about PCML, see "Program Call Markup Language" on page 362.

Example: Using ProgramCall

The following example shows how to use the ProgramCall class to use your Tier 0 device to run a program on a server:

```
// Call programs.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");

String pcmlName = "qsyrusri.pcml"; // The PCML document describing the program we want to use.
String apiName = "qsyrusri";

Hashtable parametersToSet = new Hashtable();
parametersToSet.put("qsyrusri.receiverLength", "2048");
parametersToSet.put("qsyrusri.profileName", "JOHNDOE" };

String[] parametersToGet = { "qsyrusri.receiver.userProfile",
                             "qsyrusri.receiver.previousSignonDate",
                             "qsyrusri.receiver.previousSignonTime",
                             "qsyrusri.receiver.displaySignonInfo" };

String[] valuesToGet = null;

try
{
    valuesToGet = ProgramCall.run(system, pcmlName, apiName, parametersToSet, parametersToGet);

    // Get and display the user profile.
    System.out.println("User profile: " + valuesToGet[0]);

    // Get and display the date in a readable format.
    char[] c = valuesToGet[1].toCharArray();
    System.out.println("Last Signon Date: " + c[3]+c[4]+"/"+c[5]+c[6]+"/"+c[1]+c[2] );

    // Get and display the time in a readable format.
    char[] d = valuesToGet[2].toCharArray();
    System.out.println("Last Signon Time: " + d[0]+d[1]+":"+d[2]+d[3]);

    // Get and display the signon info.
```

```
    System.out.println("Signon Info: " + valuesToGet[3] );
}
catch (MEException te)
{
    // Handle the exception.
}
catch (IOException ioe)
{
    // Handle the exception
}

// Done with the system object.
system.disconnect();
```

JdbcMe classes

The ToolboxME for iSeries classes provide JDBC support, including support for the java.sql package. The classes are meant to be used in a program that runs on a Tier 0 device.

The following sections discuss accessing and using data and describe what is in JdbcMe, including links to information about the individual JdbcMe classes.

Accessing and using data

When using a Tier0 device to access and update data, you want it to work exactly like if you were sitting at a system in your office. However, much of the development in Tier0 devices focuses on data synchronization. Using data synchronization, each Tier0 device has a copy of specific data from the main database. Periodically, users synchronize the data on each device with the main database.

Data synchronization does not work well with data that is dynamic. Working with dynamic data requires quick access to up-to-date data. Having to wait to access synchronized data is not an option for many businesses. Plus, the software and hardware demands for the servers and devices to main synchronous data can be significant.

To help solve the problems inherent in the data synchronization model, the JdbcMe classes in ToolboxME for iSeries enable you to perform live updates and access the main database, but still allow offline data storage. Your application can have access to valuable offline data without sacrificing the ability for to have live updates immediately become part of the main database. This middle ground approach provides the benefits of both the synchronous data model and the live data model.

What is in JdbcMe

By definition, a driver of any kind for a Tier0 device must be very small. The JDBC API, however, is very large. The JdbcMe classes had to be extremely small but still support enough of the JDBC interfaces so that Tier0 devices might use it to perform meaningful work.

JdbcMe classes offer the following JDBC functionality:

- The ability to insert or update data
- Transaction control and the ability to modify transaction isolation levels
- Result sets that are both scrollable and updatable
- SQL support for calls to stored procedures and drive triggers

In addition, JdbcMe classes include some unique features:

- A universal driver that enables the majority of the configuration details to be consolidated at a single point on the server side
- A standard mechanism for persisting data to offline storage

JdbcMe includes the following classes:

- JdbcMeConnection
- JdbcMeDriver
- JdbcMeException
- JdbcMeLiveResultSet
- JdbcMeOfflineData
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData
- JdbcMeStatement

ToolboxME for iSeries provides a java.sql package that follows the JDBC specification but contains only the smallest set of useful classes and methods. Providing a minimal set of sql function allows the JdbcMe classes to be small in size yet useful enough to perform common JDBC tasks.

Using ToolboxME for iSeries to connect to a database on the host server:

The JdbcMeConnection class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCConnection class. Use JdbcMeConnection to enable your Tier0 device to access DB2 Universal Database™ (UDB) databases on the host server.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see ToolboxME for iSeries requirements and installation.

Use JdbcMeDriver.getConnection() to connect to the server database. The getConnection() method takes a uniform resource locator (URL) string as an argument, the user ID, and password. The JDBC driver manager on the host server attempts to locate a driver that can connect to the database that is represented by the URL. JdbcMeDriver uses the following syntax for the URL:

```
jdbc:as400://server-name/default-schema;meserver=<server>[:port];[other properties];
```

Note: The previous syntax example is on two lines so you can easily see and print it. Normally, the URL appears on one line with no breaks or extra spaces.

You must specify a server name, or JdbcMeDriver throws an exception. The default schema is optional. If you do not specify a port, the JdbcMeDriver uses port 3470. Also, you can set a variety of JDBC properties within the URL. To set properties, use the following syntax:

```
name1=value1;name2=value2;...
```

See JDBC properties for a complete list of properties supported by JdbcMeDriver.

Examples: Using the JdbcMeDriver to connect to a server

Example: Connecting to the server database without specifying a default schema, a port, or JDBC properties

The examples specifies user ID and password as parameters on the method:

```
       // Connect to system 'mysystem'. No default schema, port or
       // properties are specified.
  Connection c  = JdbcMeDriver.getConnection("jdbc:as400://mysystem.helloworld.com;meserver=myMeServer;"
                                             "auser",
                                             "apassword");
```

Example: Connecting to the server database when specifying the schema and JDBC properties

The example specifies user ID and password as parameters on the method:

```
      // Connect to system 'mysystem'. Specify a schema and
      // two JDBC properties. Do not specify a port.
Connection c2 = JdbcMeDriver.getConnection(
   "jdbc:as400://mysystem.helloworld.com/mySchema;meserver=myMeServer;naming=system;errors=full;"
   "auser",
   "apassword");
```

Example: Connecting to the server database

The example specifies properties (including user ID and password) by using a uniform resource locator (URL):

```
      // 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");
```

Example: Disconnecting from the database

The example uses the close() method on the connecting object to disconnect from the server:

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

JdbcMeDriver class:

The JdbcMeDriver class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCDriver class. Use JdbcMeDriver in your Tier0 client application to run simple SQL statements that have no parameters and obtain ResultSets that the statements produce.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see "Downloading and setting up ToolboxME for iSeries" on page 335.

You don't explicitly register the JdbcMeDriver; instead the **driver** property you specify on the URL in the JdbcMeConnection.getConnection() method determines the driver. For example, to load the IBM Developer Kit for Java JDBC driver (called the 'native' driver), use code similar to the following:

```
Connection c = JdbcMeDriver.getConnection(
   "jdbc:as400://mysystem.myworld.com;meserver=myMeSrvr;driver=native;user=auser;password=apassword");
```

The IBM Toolbox for Java JDBC driver does not require an AS400 object as an input parameter like the other IBM Toolbox for Java classes that get data from a server. However, an AS400 object is used internally and you must explicitly provide a user ID and password. Provide the user ID and password either in the URL or by way of the parameters on the getConnection() method.

For examples of using getConnection(), see JDBCMeConnection.

Result sets:

The ToolboxME for iSeries result set classes are:

- JdbcMeLiveResultSet
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData

JdbcMeLiveResultSet and JdbcMeOfflineResultSet contain the same functionality, except that:

- JdbcMeLiveResultSet retrieves data by making a call to the database on the server
- JdbcMeOfflineResultSet retrieves data from the database on the local device

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see Downloading and setting up ToolboxME for iSeries.

JdbcMeLiveResultSet

The JdbcMeLiveResultSet class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCResultSet class. Use JdbcMeLiveResultSet in your Tier0 client application to access a table of data that is generated by running a query.

JdbcMeLiveResultSet retrieves the table rows in sequence. Within a row, you can access column values in any order. JdbcMeLiveResultSet includes methods that enable you to perform the following actions:

- Retrieve data of various types that are stored in the result set
- Move the cursor to the row you specify (previous row, current row, next row, and so on)
- Insert, update, and delete rows
- Update columns (using String and int values)
- Retrieve the ResultSetMetaData object that describes the columns in the result set

A cursor, which is an internal pointer, is used by a result set to point the row in the result set that is being accessed by the Java program. JDBC 2.0 provides additional methods for accessing specific positions within a database:

Scrollable cursor positions
absolute first last moveToCurrentRow moveToInsertRow previous relative

Scrolling capabilities

If a result set is created by executing a statement, you can move (scroll) backward (last-to-first) or forward (first-to-last) through the rows in a table.

A result set that supports this movement is called a scrollable result set. Scrollable result sets also support relative and absolute positioning. Relative positioning allows you to move to a row in the result set by specifying a position that is relative to the current row. Absolute positioning allows you to move directly to a row by specifying its position in the result set.

With JDBC 2.0, you have two additional scrolling capabilities available to use when working with the ResultSet class: scroll-insensitive and scroll-sensitive result sets.

A scroll-insensitive result set is not typically sensitive to changes that are made while it is open, while the scroll-sensitive result set is sensitive to changes. The IBM Toolbox for Java JDBC driver does not support scroll-insensitive result sets.

Updatable result sets

In your application, you can use result sets that use either read-only concurrency (no updates can be made to the data) or updatable concurrency (allows updates to the data and may use database write locks to control access to the same data item by different transactions). In an updatable result set, rows can be updated, inserted, and deleted.

See Method Summary for a complete listing of the update methods available in JdbcMeResultSet.

Example: Updatable result sets

The following example shows how to use a result set that allows updates to the data (update concurrency) and allows changes to be made to the result set while it is open (scroll sensitive).

```
     // 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 class

The JdbcMeOfflineResultSet class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCResultSet class. Use JdbcMeOfflineResultSet in in your Tier0 client application to access a table of data that is generated by running a query.

Use the JdbcMeOfflineResultSet class to work with data that resides on your Tier0 device. The data that resides on the device might already reside there or you might have put it there by calling JdbcMeStatement.executeToOfflineData() method. the executeToOfflineData() method downloads and stores to the device all of the data that satisfies the query. You can then use JdbcMeOfflineResultSet class to access the stored data.

JdbcMeOfflineResultSet includes methods that enable you to perform the following actions:

- Retrieve data of various types that are stored in the result set
- Move the cursor to the row you specify (previous row, current row, next row, and so on)
- Insert, update, and delete rows

- Update columns (using String and int values)
- Retrieve the ResultSetMetaData object that describes the columns in the result set

You can provide the ability to synchronize the local device database with the database on the iSeries server by using the functions present in the JdbcMe classes.

JdbcMeResultSetMetaData class

The JdbcMeResultSetMetaData class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCResultSetMetaData class. Use JdbcMeResultSetMetaData in your Tier0 client application to determine the types and properties of the columns in a JdbcMeLiveResultSet or JdbcMeOfflineResultSet.

The following example shows how to use the JdbcMeResultSetMetaData class:

```
     // Connect to the server.
Connection c = JdbcMeDriver.getConnection(
   "jdbc:as400://mySystem;meserver=myMeServer;user=auser;password=apassword");

     // Create a Statement object.
Statement s = c.createStatement();

     // Run a query. The result is placed in a ResultSet object.
JdbcMeLiveResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE");

     // Iterate through the rows of the ResultSet.
while (rs.next ())
{

         // Get the values from the ResultSet. The first value is
         // a string, and the second value is an integer.
    String name = rs.getString("NAME");
    int id = rs.getInt("ID");

    System.out.println("Name = " + name);
    System.out.println("ID = " + id);
}

     // Close the Statement and the Connection.
s.close();
c.close();
```

JdbcMeOfflineData class:

The JdbcMeOfflineData class is an offline data repository meant to be used on a Tier0 device. The repository is generic, regardless of the profile and Java virtual machine that you are using. For more information, see ToolboxME for iSeries Concepts.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see Downloading and setting up ToolboxME for iSeries.

The JdbcMeOfflineData class provides methods that enable you to perform the following functions:

- Create an offline data repository
- Open an existing repository
- Get the number of records in the repository
- Get and delete individual records
- Update records (Robb: the set() method, right?)
- Add a record to the end of the repository
- Close the repository

For an example of using the JdbcMeOfflineData class, see the following example:

"Example: Using ToolboxME for iSeries, MIDP, and IBM Toolbox for Java" on page 672

JdbcMeStatement class:

The JdbcMeStatement class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCStatement class. Use JdbcMeStatement in your Tier0 client application to run simple SQL statements that have no parameters and obtain ResultSets that the statements produce.

Note: To use ToolboxMe for iSeries classes, you must separately download and set up the ToolboxME for iSeries component. For more information, see Downloading and setting up ToolboxME for iSeries.

Use JdbcMeConnection.createStatement() to create new Statement objects.

The following example shows how to use a JdbcMeStatement object:

```
     // 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();
```

Creating and running a ToolboxME for iSeries program

This information will enable you to edit, compile, and run the example ToolboxME for iSeries program.

You can also use this information as a general guide for creating, testing, and running the ToolboxME for iSeries working examples and your own ToolboxME for iSeries applications.

The example program uses the K Virtual Machine (KVM) and allows the user to perform any JDBC query. The user can then perform JDBC actions (next, previous, close, commit, and rollback) against the result of the query.

Before you begin creating any of the ToolboxME for iSeries examples, make sure that your environment meets the ToolboxME for iSeries requirements.

Creating the ToolboxME for iSeries example

To create the ToolboxME for iSeries example program for your Tier0 device, complete the following steps:

1. Copy the Java code for the ToolboxME for iSeries example, called JdbcDemo.java.
2. In your chosen text or Java editor, change the portions of the code as indicated in the program comments and save the file with the name JdbcDemo.java.

Note: Consider using a wireless application development tool, which makes it easier to complete the remaining steps. Some wireless application development tools may compile, preverify, and build your program in a single step, then automatically run it in an emulator.

3. Compile JdbcDemo.java, making sure you point to the .jar file that contains the KVM classes.
4. Preverify the executable file, either by using your wireless application development tool or by using the Java preverify command.
5. Build the appropriate type of executable file for the operating system of your Tier0 device. For example, for the Palm OS, you build a file called JdbcDemo.prc.
6. Test the program. If you have installed an emulator, you can test the program and see what it will look like by running it in the emulator.

   Note: If you test the program on your wireless device and you do not use a wireless application development tool, make sure that you preinstall your chosen Java virtual machine or MIDP on the device.

See ToolboxME for iSeries concepts for related information about concepts, wireless application development tools, and emulators.

Running the ToolboxME for iSeries example

To run the ToolboxME for iSeries example program on your Tier0 device, complete the following tasks:

- Load the executable file to the device, using the instructions provided by your Tier0 device manufacturer.
- Start the MEServer
- Run the JdbcDemo program on your Tier0 device by clicking the JdbcDemo icon.

ToolboxME for iSeries example: JdbcDemo.java

To create this example as a working ToolboxME for iSeries program, you need to copy the following .java file into a text or Java editor, make a few changes, then compile it.

To copy the source code, simply use your mouse to select all the Java code below, then right-click and select **Copy**. To paste the code into your editor, create a blank document in the editor, right-click the blank document and select **Paste**. Make sure to save the new document with the name JdbcDemo.java.

After you create the .java file, return to the instructions for creating and running the example program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME for iSeries example. This program demonstrates how your wireless
// device can connect to an iSeries 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 iSeries 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 working examples

The following ToolboxMe for iSeries working examples illustrate ways to use ToolboxME for iSeries with the Mobile Information Device Profile (MIDP).

Use the following links to view selected example source files or to download all the example source files required to build the working example wireless applications:

"Example: Using ToolboxME for iSeries, MIDP, and JDBC" on page 664

"Example: Using ToolboxME for iSeries, MIDP, and IBM Toolbox for Java" on page 672

"Downloading the ToolboxME for iSeries examples"

For more information about how to build a ToolboxME for iSeries application, see "Creating and running a ToolboxME for iSeries program" on page 347.

For more information about MIDP, see "Mobile Information Device Profile (MIDP)" on page 336.

Downloading the ToolboxME for iSeries examples

To build the ToolboxME for iSeries examples into working wireless applications, you need all the source files and additional instructions.

To download and build the examples, complete the following steps:

1. Download the source files (microsamples.zip).

2. Unzip microsamples.zip into a directory you create for that purpose.
3. Use the instructions provided in "Creating and running a ToolboxME for iSeries program" on page 347 to help you build the example wireless applications.

Before you begin compiling the source and building the executable files for your Tier0 device, see the following for more information:

- "ToolboxME for iSeries requirements" on page 8
- "Downloading and setting up ToolboxME for iSeries" on page 335

Extensible Markup Language components

IBM Toolbox for Java includes several Extensible Markup Language (XML) components, including an XML parser.

The XML components make it easier to perform a variety of tasks:

- Creating graphical user interfaces
- Calling programs on your iSeries server and retrieving the results
- Specifying data formats on your iSeries server

Program Call Markup Language

Program Call Markup Language (PCML) is a tag language that helps you call server programs, with less Java code.

PCML is based upon the Extensible Markup Language (XML), a tag syntax you use to describe the input and output parameters for server programs. PCML enables you to define tags that fully describe server programs called by your Java application.

Note: If you are interested in or are already using PCML, consider using Extensible Program Call Markup Language (XPCML). XPCML enhances the functionality and usability of PCML by offering support for XML schemas. For more information about IBM Toolbox for Java XML components, including XPCML, see Extensible Markup Language components.

A huge benefit of PCML is that it allows you to write less code. Ordinarily, extra code is needed to connect, retrieve, and translate data between a server and IBM Toolbox for Java objects. However, by using PCML, your calls to the server with the IBM Toolbox for Java classes are automatically handled. PCML class objects are generated from the PCML tags and help minimize the amount of code you need to write in order to call server programs from your application.

Although PCML was designed to support distributed program calls to server program objects from a client Java platform, you can also use PCML to make calls to a server program from within the server environment.

Refer to the following pages for more information about using PCML:

- "Building iSeries program calls with PCML"
- "PCML syntax" on page 367
- "Examples: Program Call Markup Language (PCML)" on page 587

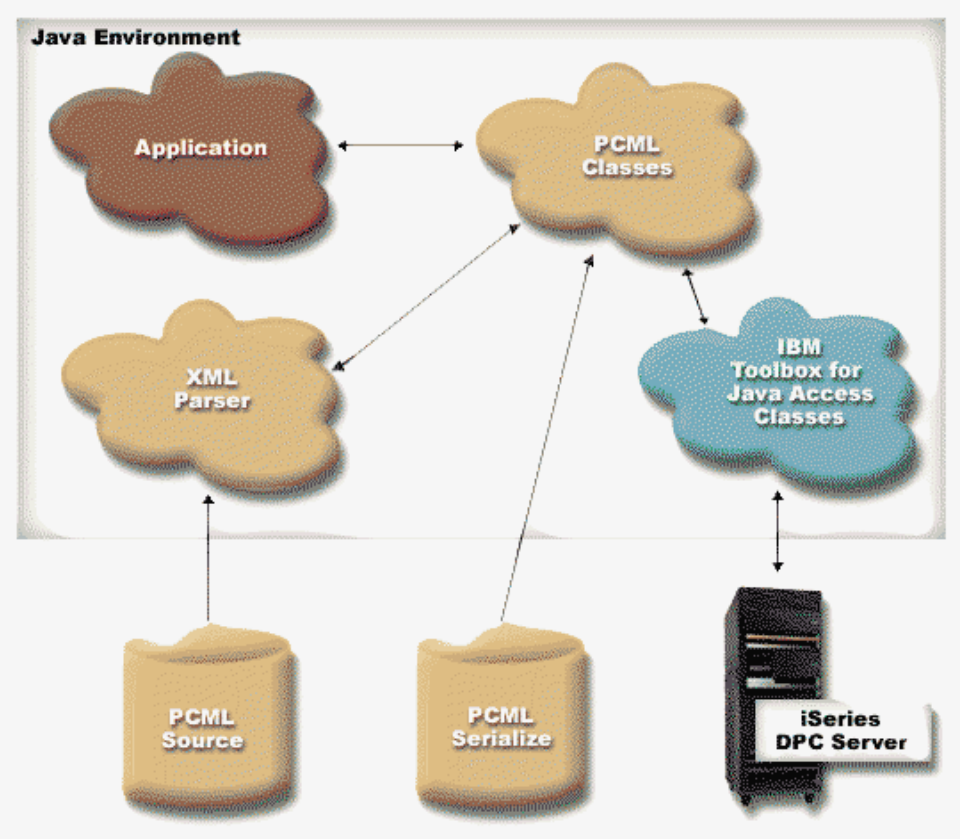
Building iSeries program calls with PCML

To build iSeries program calls with PCML, you must start by creating a Java application and a PCML source file.

Depending on your design process, you must write one or more PCML source files where you describe the interfaces to the iSeries programs that will be called by your Java application. Refer to PCML syntax for a detailed description of the language.

Then your Java application interacts with the PCML classes (in this case, the ProgramCallDocument class). The ProgramCallDocument class uses your PCML source file to pass information between your Java application and the iSeries programs. Figure 1 illustrates how Java applications interact with the PCML classes.

Figure 1. Making program calls to the server using PCML.



When your application constructs the ProgramCallDocument object, the XML parser reads and parses the PCML source file. For more information about using an XML parser with IBM Toolbox for Java, see XML parser and XSLT processor.

After the ProgramCallDocument class has been created, the application program uses the ProgramCallDocument class's methods to retrieve the necessary information from the server through the iSeries distributed program call (DPC) server.

To improve run-time performance, the ProgramCallDocument class can be serialized during your product build time. The ProgramCallDocument is then constructed using the serialized file. In this case, the XML parser is not used at run-time. Refer to Using serialized PCML files.

Using PCML source files

Your Java application uses PCML by constructing a ProgramCallDocument object with a reference to the PCML source file. The ProgramCallDocument object considers the PCML source file to be a Java resource. The java application finds the PCML source file by using the Java CLASSPATH

The following Java code constructs a ProgramCallDocument object:

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "myPcmlDoc");
```

The ProgramCallDocument object will look for your PCML source in a file called myPcmlDoc.pcml. Notice that the .pcml extension is not specified on the constructor.

If you are developing a Java application in a Java package, you can package-qualify the name of the PCML resource:

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.company.package.myPcmlDoc");
```

Using serialized PCML files

To improve run-time performance, you can use a serialized PCML file. A serialized PCML file contains serialized Java objects representing the PCML. The objects that are serialized are the same objects that are created when you construct the ProgramCallDocument from a source file as described above.

Using serialized PCML files improves performance because the XML parser is not needed at run-time to process the PCML tags.

The PCML can be serialized using either of the following methods:

- From the command line:

  ```
  java com.ibm.as400.data.ProgramCallDocument -serialize mypcml
  ```

  This method is helpful for having batch processes to build your application.
- From within a Java program:

  ```
  ProgramCallDocument pcmlDoc; // Initialized elsewhere
  pcmlDoc.serialize();
  ```

If your PCML is in a source file named myDoc.pcml, the result of serialization is a file named myDoc.pcml.ser.

PCML source files vs. serialized PCML files

Consider the following code to construct a ProgramCallDocument:

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.mycompany.mypackage.myPcmlDoc");
```

The ProgramCallDocument constructor will first try to find a serialized PCML file named myPcmlDoc.pcml.ser in the com.mycompany.mypackage package in the Java CLASSPATH. If a serialized PCML file does not exist, the constructor will then try to find a PCML source file named myPcmlDoc.pcml in the com.mycompany.mypackage package in the Java CLASSPATH. If a PCML source file does not exist, an exception is thrown.

Qualified names

Your Java application uses ProgramCallDocument.setValue() to set input values for the iSeries program being called. Likewise, your application uses ProgramCallDocument.getValue() to retrieve output values from the iSeries program.

When accessing values from the ProgramCallDocument class, you must specify the fully qualified name of the document element or <**data**> tag. The qualified name is a concatenation of the names of all the containing tags with each name separated by a period.

For example, given the following PCML source, the qualified name for the "**nbrPolygons**" item is "**polytest.parm1.nbrPolygons**". The qualified name for accessing the "**x**" value for one of the points in one of the polygons is "**polytest.parm1.polygon.point.x**".

If any one of the elements needed to make the qualified name is unnamed, all descendants of that element do not have a qualified name. Any elements that do not have a qualified name cannot be accessed from your Java program.

```
<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>
```

Accessing data in arrays

Any <**data**> or <**struct**> element can be defined as an array using the **count** attribute. Or, a <**data**> or <**struct**> element can be contained within another <**struct**> element that is defined as an array.

Furthermore, a <**data**> or <**struct**> element can be in a multidimensional array if more than one containing element has a **count** attribute specified.

In order for your application to set or get values defined as an array or defined within an array, you must specify the array index for each dimension of the array. The array indices are passed as an array of **int** values. Given the source for the array of polygons shown above, the following Java code can be used to retrieve the information about the polygons:

```
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);
    }
}
```

Debugging

When you use PCML to call programs with complex data structures, it is easy to have errors in your PCML that result in exceptions from the ProgramCallDocument class. If the errors are related to incorrectly describing offsets and lengths of data, the exceptions can be difficult to debug.

Use the following method from the Trace class to turn on PCML tracing:

```
Trace.setTraceOn(true);      // Turn on tracing function.
Trace.setTracePCMLOn(true);  // Turn on PCML tracing.
```

Note: All public methods in the PcmlMessageLog class, including tracing, were deprecated in V5R2.

The Trace setFileName() method enables you to send the following types of information to specific log files or, by default, to System.out:

- A dump of the hexadecimal data being transferred between the Java application and the iSeries program. This shows the program input parameters after character data is converted to EBCDIC and integers are converted to big-endian. It also shows the output parameters before they are converted to the Java environment.

  The data is shown in a typical hexadecimal dump format with hexadecimal digits on the left and a character interpretation on the right. The following is an example of this dump format. (The following example was altered to allow for width restrictions)

  ```
  qgyolobj[6]
  Offset : 0....... 4....... 8....... C....... 0....... 4....... 8....... C.......
           0...4...8...C...0...4...8...C...
       0 : 5CE4E2D9 D7D9C640 4040
           **USRPRF                        *
  ```

  In the above example, the dump shows the seventh parameter has 10 bytes of data set to ″*USRPRF ″.

- For output parameters, following the hexadecimal dump is a description of how the data has been interpreted for the document. (The following example was altered to allow for width restrictions)

```
/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
```

The above messages can be very helpful in diagnosing cases where the output data coming from the iSeries program does not match the PCML source. This can easily occur when you are using dynamic lengths and offsets.

Long description of Figure 1: Making program calls to the server using PCML (rzahh503.gif):

found in IBM Toolbox for Java: PCML Process

This image illustrates how Java applications can interact with the PCML classes.

Description

The image is divided into two areas: an upper portion that represents the Java environment and a lower portion that represents the nonJava part of the PCML process.

- The Java environment (upper portion) includes four shapes labeled ″Application,″ ″PCML Classes,″ ″IBM Toolbox for Java Access Classes,″ and ″XML Parser.″
- The nonJava part of the process (lower portion) includes two shapes labeled ″PCML Source,″ and ″PCML Serialize,″ and an image of an iSeries server labeled ″iSeries DPC Server.″
- Arrows that point one or both directions connect the shapes. An arrow that points to both shapes means that the two shapes interact with one another. An arrow that points in only one direction means that the shape pointed to uses the other shape in some way.

The Java application interacts with the PCML classes. In this example, the application creates a ProgramCallDocument object.

When the ProgramCallDocument is constructed, one of two things happens:

- The XML parser parses the PCML source files and passes the information to the PCML classes. The PCML source files describe the interfaces to the iSeries programs called by your Java application.
- Serialized PCML information is passed to the PCML classes. Using serialized PCML improves run-time performance because the PCML has already been parsed. If you choose to serialize your PCML source, you must do so when you build your application.

The PCML classes also interact with the IBM Toolbox for Java classes, which in this example use the iSeries distributed program call server to retrieve information from the server.

These actions and interactions enable information to pass between the Java application and iSeries programs.

PCML syntax

PCML consists of the following tags, each of which has its own attribute tags.

- The program tag begins and ends code that describes one program
- The struct tag defines a named structure which can be specified as an argument to a program or as a field within another named structure. A structure tag contains a data or a structure tag for each field in the structure.
- The data tag defines a field within a program or structure.

In the following example the PCML syntax describes one program with one category of data and some isolated data.

```
<program>

    <struct>
        <data> </data>
    </struct>

    <data> </data>

</program>
```

PCML program tag:

The PCML program tag can be expanded with the following elements.

```
<program name="name"
    [ entrypoint="entry-point-name" ]
    [ epccsid="ccsid" ]
    [ path="path-name" ]
    [ parseorder="name-list" ]
    [ returnvalue="{ void | integer }" ]
    [ threadsafe="{ true | false }" ]>
</program>
```

The following table lists the program tag attributes. Each entry includes the attribute name, the possible valid values, and a description of the attribute.

Attribute	Value	Description
entrypoint=	*entry-point-name*	Specifies the name of the entry point within a service program object that is the target of this program call.
epccsid=	*ccsid*	Specifies the CCSID of the entry point within a service program. For more information, see the service program entry notes in the ServiceProgramCall javadoc.
name=	*name*	Specifies the name of the program.
path=	*path-name*	Specifies the path to the program object. The default value is to assume the program is in the QSYS library. The path must be a valid integrated file system path name to a *PGM or *SRVPGM object. If a *SRVPGM object is called, the entrypoint attribute must be specified to indicate the name of the entrypoint to be called. If the entrypoint attribute is not specified, the default value for this attribute is assumed to be a *PGM object from the QSYS library. If the entrypoint attribute is specified, the default value for this attribute is assumed to be a *SRVPGM object in the QSYS library. The path name must be specified as all uppercase characters. Do not use the **path** attribute when the application needs to set the path at run time, for example, when a user specifies what library is used for the install. In this case, use the ProgramCallDocument.setPath() method.

Attribute	Value	Description
parseorder=	*name-list*	Specifies the order in which output parameters will be processed. The value specified is a blank separated list of parameter names in the order in which the parameters are to be processed. The names in the list must be identical to the names specified on the **name** attribute of tags belonging to the **<program>**. The default value is to process output parameters in the order the tags appear in the document. Some programs return information in one parameter that describes information in a previous parameter. For example, assume a program returns an array of structures in the first parameter and the number of entries in the array in the second parameter. In this case, the second parameter must be processed in order for the ProgramCallDocument to determine the number of structures to process in the first parameter.
returnvalue=	*void* The program does not return a value. *integer* The program returns a 4-byte signed integer.	Specifies the type of value, if any, that is returned from a service program call. This attribute is not allowed for *PGM object calls.
threadsafe=	*true* The program is considered to be thread-safe. *false* The program is not thread-safe.	When you call a Java program and an iSeries program that are on the same server, use this property to specify whether you want to call the iSeries program in the same job and on the same thread as the Java program. If you know your program is thread-safe, setting the property to *true* results in better performance. To keep the environment safe, the default is to call programs in separate server jobs. The default value is *false*.

PCML struct tag:

The PCML struct tag can be expanded with the following elements.

```
<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>
```

The following table lists the struct tag attributes. Each entry includes the attribute name, the possible valid values, and a description of the attribute.

Attribute	Value	Description
name=	*name*	Specifies the name of the **<struct>** element
count=	*number* where *number* defines a fixed, never-changing sized array. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of elements in the array. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies that the element is an array and identifies the number of entries in the array. If this attribute is omitted, the element is not defined as an array, although it may be contained within another element that is defined as an array.
maxvrm=	*version-string*	Specifies the highest version of i5/OS on which the element exists. If the version of i5/OS is greater than the version specified on the attribute, the element and its children, if any exist, will not be processed during a call to a program. The **maxvrm** element is helpful for defining program interfaces which differ between releases of i5/OS. The syntax of the version string must be ″VvRrMm,″ where the capitals letters ″V,″ ″R,″ and ″M″ are literal characters and ″v,″ ″r,″ and ″m″ are one or more digits representing the version, release and modification level. The value for ″v″ must be from 1 to 255 inclusively. The value for ″r″ and ″m″ must be from 0 to 255, inclusively.

Attribute	Value	Description
minvrm=	*version-string*	Specifies the lowest version of i5/OS on which this element exists. If the version of i5/OS is less than the version specified on this attribute, this element and its children, if any exist, will not be processed during a call to a program. This attribute is helpful for defining program interfaces which differ between releases of i5/OS. The syntax of the version string must be ″VvRrMm,″ where the capitals letters ″V,″ ″R,″ and ″M″ are literal characters and ″v,″ ″r,″ and ″m″ are one or more digits representing the version, release and modification level. The value for ″v″ must be from 1 to 255, inclusively. The value for ″r″ and ″m″ must be from 0 to 255, inclusively.
offset=	*number* where *number* defines a fixed, never-changing offset. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the offset to the element. The **data-name** specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***″int″*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the offset to the **<struct>** element within an output parameter. Some programs return information with a fixed structure followed by one or more variable length fields or structures. In this case, the location of a variable length element is typically specified as an offset or displacement within the parameter. The **offset** attribute is used to describe the offset to this **<struct>** element. **Offset** is used in conjunction with the **offsetfrom** attribute. If the **offsetfrom** attribute is not specified, the base location for the offset specified on the **offset** attribute is the parent of the element. See Specifying Offsets for more information on how to use the **offset** and **offsetfrom** attributes. The **offset** and **offsetfrom** attributes are only used to process output data from a program. These attributes do not control the offset or displacement of input data. If the attribute is omitted, the location of the data for the element is immediately following the preceding element in the parameter, if any.

Attribute	Value	Description
offsetfrom=	*number* where *number* defines a fixed, never-changing base location. A *number* attribute is most typically used to specify **number=***"0"* indicating that the offset is an absolute offset from the beginning of the parameter. *data-name* where *data-name* defines the name of a **<data>** element to be used as a base location for the offset. The element name specified must be the parent or an ancestor of this element. The value from the **offset** attribute will be relative to the location of the element specified on this attribute. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference an ancestor of this element. See Resolving Relative Names for more information on how relative names are resolved. *struct-name* where *struct-name* defines the name of a **<struct>** element to be used as a base location for the offset. The element name specified must be the parent or an ancestor of this element. The value from the **offset** attribute will be relative to the location of the element specified on this attribute. The *struct-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference an ancestor of this element. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the base location from which the **offset** attribute is relative. If the **offsetfrom** attribute is not specified, the base location for the offset specified on the **offset** attribute is the parent of this element. See Specifying Offsets for more information on how to use the **offset** and **offsetfrom** attributes. The **offset** and **offsetfrom** attributes are only used to process output data from a program. These attributes do not control the offset or displacement of input data.

Attribute	Value	Description
outputsize=	*number* where *number* defines a fixed,never-changing number of bytes to reserve. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of bytes to reserve for output data. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the number of bytes to reserve for output data for the element. For output parameters which are variable in length, the **outputsize** attribute is needed to specify how many bytes must be reserved for data to be returned from the server program. **Outputsize** can be specified on all variable length fields and variable sized arrays, or it can be specified for an entire parameter that contains one or more variable length fields. **Outputsize** is not necessary and must not be specified for fixed-size output parameters. The value specified on the attribute is used as the total size for the element including all children of the element. Therefore, the **outputsize** attribute is ignored on any children or descendants of the element. If the attribute is omitted, the number of bytes to reserve for output data is determined at runtime by adding the number of bytes to reserve for all of the children of the **<struct>** element.
usage=	*inherit*	Usage is inherited from the parent element. If the structure does not have a parent, usage is assumed to be **inputoutput**.
	input	The structure is an input value to the host program. For character and numeric types, the appropriate conversion is performed.
	output	The structure is an output value from the host program. For character and numeric types, the appropriate conversion is performed.
	inputoutput	The structure is both and input and an output value.

Specifying offsets

Some programs return information with a fixed structure followed by one or more variable length fields or structures. In this case, the location of a variable length element is typically specified as an offset or displacement within the parameter.

An offset is the distance in bytes from a the beginning of the parameters to the beginning of a field or structure. A displacement is the distance in bytes from the beginning of one structure to the beginning of another structure.

For offsets, since the distance is from the beginning of the parameter, specify **offsetfrom=″0″**. The following is an example of an offset from the beginning of the parameter:

```
<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>
```

For displacements, since the distance is from the beginning of another structure, you specify the name of the structure to which the offset is relative. The following is an example of an displacement from the beginning of a named structure:

```
<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 tag:

The PCML data tag can have the following attributes.

Attributes enclosed in brackets, [], indicate that the attribute is optional. If you specify an optional attribute, do not include the brackets in your source. Some attribute values are shown as a list of choices enclosed in braces, {}, with possible choices separated by vertical bars, |. When you specify one of these attributes, do not include the braces in your source and only specify one of the choices shown.

```
<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>
```

The following table lists the data tag attributes. Each entry includes the attribute name, the possible valid values, and a description of the attribute.

Attribute	Value	Description
type=	*char* where *char* indicates a character value. A *char* data value is returned as a *java.lang.String*. For more information, see the *char* values for length. *int* where *int* is an integer value. An *int* data value is returned as a *java.lang.Long*. For more information, see the *int* values for length and precision. *packed* where *packed* is a packed decimal value. A *packed* data value is returned as a *java.math.BigDecimal*. For more information, see the *packed* values for length and precision. *zoned* where *zoned* is a zoned decimal value. A *zoned* data value is returned as a *java.math.BigDecimal*. For more information, see the *zoned* values for length and precision. *float* where *float* is a floating point value. The **length** attribute specifies the number of bytes, "4" or "8". A 4-byte integer is returned as a *java.lang.Float*. An 8-byte integer is returned as a *java.lang.Double*. For more information, see the *float* values for length. *byte* where *byte* is a byte value. No conversion is performed on the data. A *byte* data value is returned as an array of *byte* values (*byte[]*). For more information, see the *byte* values for length. *struct* where *struct* specifies the name of the **<struct>** element. A *struct* allows you to define a structure once and reuse it multiple times within the document. When you **type=***"struct"*, it is as if the structure specified appeared at this location in the document. A *struct* does not allow for a length value and has no value for precision.	Indicates the type of data being used (character, integer, packed, zoned, floating point, byte, or struct). Values for the length and precision attributes are different for different data types. For more information, see the Values for length and precision.

Attribute	Value	Description
bidistringtype=	*DEFAULT* where DEFAULT is the default string type for non-bidirectional data (LTR). *ST4* where *ST4* is String Type 4. *ST5* where *ST5* is String Type 5. *ST6* where *ST6* is String Type 6. *ST7* where *ST7* is String Type 7. *ST8* where *ST8* is String Type 8. *ST9* where *ST9* is String Type 9. *ST10* where *ST10* is String Type 10. *ST11* where *ST11* is String Type 11.	Specifies the bidirectional string type for **<data>** elements with **type=***"char"*. If this attribute is omitted, string type for this element is implied by the CCSID (whether explicitly specified or the default CCSID of the host environment). String types are defined in the javadoc for the BidiStringType class.
ccsid=	*number* where *number* defines a fixed, never-changing CCSID. *data-name* where *data-name* defines the name that will contain, at runtime, the CCSID of the character data. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the host Coded Character Set ID (CCSID) for character data for the **<data>** element. The **ccsid** attribute can be specified only for **<data>** elements with **type=***"char"*. If this attribute is omitted, character data for this element is assumed to be in the default CCSID of the host environment.
chartype=	*onebyte* where *onebyte* specifies the size of each character. *twobyte* where *twobyte* specifies the size of each character. When using *chartype*, the **length=***"number"* attribute specifies the number of characters, not the number of bytes.	Specifies the size of each character.
count=	*number* where *number* defines a fixed, never-changing number of elements in a sized array. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of elements in the array. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies that the element is an array and identifies the number of entries in the array. If the *count* attribute is omitted, the element is not defined as an array, although it may be contained within another element that is defined as an array.

Attribute	Value	Description
init=	*string*	Specifies an initial value for the **<data>** element. The *init* value is used if an initial value is not explicitly set by the application program when **<data>** elements with **usage=***"input"* or **usage=***"inputoutput"* are used. The initial value specified is used to initialize scalar values. If the element is defined as an array or is contained within a structure defined as an array, the initial value specified is used as an initial value for all entries in the array.
length=	*number* where *number* defines the number of bytes that the data requires. However, when using the *chartype* attribute, *number* specifies the number of characters, not the number of bytes. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the length. A *data-name* can be specified only for **<data>** elements with **type=***"char"* or **type=***"byte"*. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the length of the data element. Usage of this attribute varies depending on the data type. For more information, see the Values for length and precision.
maxvrm=	*version-string*	Specifies the highest version of iSeries on which this element exists. If the iSeries version is greater than the version specified on this attribute, this element and its children, if any exist, will not be processed during a call to a program. This attribute is helpful for defining program interfaces which differ between releases of iSeries. The syntax of the version string must be ″VvRrMm″, where the capitals letters ″V,″ ″R,″ and ″M″ are literal characters and ″v,″ ″r,″ and ″m″ are one or more digits representing the version, release and modification level. The value for ″v″ must be from 1 to 255 inclusively. The value for ″r″ and ″m″ must be from 0 to 255, inclusively.

Attribute	Value	Description
minvrm=	*version-string*	Specifies the lowest version of iSeries on which this element exists. If the iSeries version is less than the version specified on this attribute, this element and its children, if any exist, will not be processed during a call to a program. This attribute is helpful for defining program interfaces which differ between releases of iSeries. The syntax of the version string must be ″VvRrMm,″ where the capitals letters ″V,″ ″R,″ and ″M″ are literal characters and ″v,″ ″r,″ and ″m″ are one or more digits representing the version, release and modification level. The value for ″v″ must be from 1 to 255 inclusively. The value for ″r″ and ″m″ must be from 0 to 255, inclusively.
name=	*name*	Specifies the name of the **<data>** element.
offset=	*number* where *number* defines a fixed, never-changing offset. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the offset to this element. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***″int″*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the offset to the **<data>** element within an output parameter. Some programs return information with a fixed structure followed by one or more variable length fields or structures. In this case, the location of a variable length element is typically specified as an offset or displacement within the parameter. An **offset** attribute is used in conjunction with the **offsetfrom** attribute. If the **offsetfrom** attribute is not specified, the base location for the offset specified on the **offset** attribute is the parent of this element. See Specifying Offsets for more information on how to use the **offset** and **offsetfrom** attributes. The **offset** and **offsetfrom** attributes are only used to process output data from a program. These attributes do not control the offset or displacement of input data. If this attribute is omitted, the location of the data for this element is immediately following the preceding element in the parameter, if any.

Attribute	Value	Description
offsetfrom=	*number* where *number* defines a fixed, never-changing base location. *Number* is most typically used to specify **number=***"0"* indicating that the offset is an absolute offset from the beginning of the parameter. *data-name* where *data-name* defines the name of a **<data>** element used as a base location for the offset. The element name specified must be the parent or an ancestor of this element. The value from the **offset** attribute will be relative to the location of the element specified on this attribute. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference an ancestor of this element. See Resolving Relative Names for more information on how relative names are resolved. *struct-name* where *struct-name* defines the name of a **<struct>** element used as a base location for the offset. The element name specified must be the parent or an ancestor of this element. The value from the **offset** attribute will be relative to the location of the element specified on this attribute. The *struct-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference an ancestor of this element. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the base location from which the **offset** attribute is relative. If the **offsetfrom** attribute is not specified, the base location for the offset specified on the **offset** attribute is the parent of this element. See Specifying Offsets for more information on how to use the **offset** and **offsetfrom** attributes. The **offset** and **offsetfrom** attributes are only used to process output data from a program. These attributes do not control the offset or displacement of input data.

Attribute	Value	Description
outputsize=	*number* where a *number* defines a fixed, never-changing number of bytes to reserve. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of bytes to reserve for output data. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information on how relative names are resolved.	Specifies the number of bytes to reserve for output data for the element. For output parameters which are variable in length, the **outputsize** attribute is needed to specify how many bytes must be reserved for data to be returned from the iSeries program. An **outputsize** attribute can be specified on all variable length fields and variable sized arrays, or it can be specified for an entire parameter that contains one or more variable length fields. **Outputsize** is not necessary and must not be specified for fixed-size output parameters. The value specified on this attribute is used as the total size for the element including all the children of the element. Therefore, the **outputsize** attribute is ignored on any children or descendants of the element. If **outputsize** is omitted, the number of bytes to reserve for output data is determined at runtime by adding the number of bytes to reserve for all of the children of the **<struct>** element.
passby=	*reference* where *reference* indicates that the parameter will be passed by reference. When the program is called, the program will be passed a pointer to the parameter value. *value* where *value* indicates an integer value. This value is allowed only when **type=** *"int"* and **length=***"4"* is specified.	Specifies whether the parameter is passed by reference or passed by value. This attribute is allowed only when this element is a child of a **<program>** element defining a service program call.
precision=	*number*	Specifies the number of bytes of precision for some numeric data types. For more information, see the Values for length and precision.
struct=	*name*	Specifies the name of a **<struct>** element for the **<data>** element. A **struct** attribute can be specified only for **<data>** elements with **type=***"struct"*.

Attribute	Value	Description
trim=	*right* where *right* is the default behavior that means to trim trailing white spaces. *left* where *left* means to trim preceding white spaces. *both* where *both* means to trim both preceding and trailing white spaces. *none* where *none* means that white spaces are not trimmed.	Specifies how to trim white space from character data.
usage=	*inherit*	Usage is inherited from the parent element. If the structure does not have a parent, usage is assumed to be *inputoutput*.
	input	Defines an input value to the host program. For character and numeric types, the appropriate conversion is performed.
	output	Defines an output value from the host program. For character and numeric types, the appropriate conversion is performed.
	inputoutput	Defines both and input and an output value.

Specifying offsets

Some programs return information with a fixed structure followed by one or more variable length fields or structures. In this case, the location of a variable length element is typically specified as an offset or displacement within the parameter.

An offset is the distance in bytes from the beginning of the parameters to the beginnings of a field or structure. A displacement is the distance in bytes from the beginning of one structure to the beginning of another structure.

For offsets, since the distance is from the beginning of the parameter, you must specify **offsetfrom=″0″**. The following is an example of an offset from the beginning of the parameter:

```
<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>
```

For displacements, since the distance is from the beginning of another structure, you specify the name of the structure to which the offset is relative. The following is an example of an displacement from the beginning of a named structure:

```
<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>
```

Values for length and precision:

Values for the length and precision attributes are different for different data types.

The following table lists each data type with a description of the possible values for length and precision.

Data type	Length	Precision
type="char"	The number of bytes of data for this element, which is not necessarily the number of characters. You must specify either a literal *number* or a *data-name*.	Not applicable
type="int"	The number of bytes of data for this element: 2, 4, or 8. You must specify a literal *number*.	Indicates the number of bits of precision and whether the integer is signed or unsigned: • For **length**="2" – Use **precision**="15" for a signed 2-byte integer. This is the default value – Use **precision**="16" for an unsigned 2-byte integer • For **length**="4" – Use **precision**="31" for a signed 4-byte integer – Use **precision**="32" for an unsigned 4-byte integer • For **length**="8" use **precision**="63" for a signed 8-byte integer
type="packed" or "zoned"	The number of numeric digits of data for this element. You must specify a literal *number*.	The number of decimal digits for the element. This number must be greater than or equal to zero and less than or equal to the total number of digits specified on the **length** attribute.
type="float"	The number of bytes, 4 or 8, of data for this element. You must specify a literal *number*.	Not applicable
type="byte"	The number of bytes of data for this element. You must specify either a literal *number* or *data-name*.	Not applicable

Data type	Length	Precision
type="struct"	Not allowed.	Not applicable

Resolving relative names

Several attributes allow you to specify the name of another element, or tag, within the document as the attribute value. The name specified can be a name that is relative to the current tag.

Names are resolved by seeing if the name can be resolved as a child or descendent of the tag containing the current tag. If the name cannot be resolved at this level, the search continues with the next highest containing tag. This resolution must eventually result in a match of a tag that is contained by either the **<pcml>** tag or the <**rfml**> tag, in which case the name is considered to be an absolute name, not a relative name.

Here is an example using PCML:

```
<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>
```

Here is an example using RFML:

```
<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>
```

Record Format Markup Language

The Record Format Markup Language (RFML) is an XML extension for specifying record formats.

The IBM Toolbox for Java RFML component enables your Java applications to use RFML documents to specify and manipulate fields within certain kinds of records.

RFML documents, called RFML source files, represent a useful subset of the data description specification (DDS) data types defined for physical and logical files on iSeries servers. You can use RFML documents to manage the information in the following:

- File records
- Data queue entries
- User spaces
- Arbitrary data buffers

Note: For more information about using DDS to describe data attributes, see the DDS Reference.

RFML closely resembles Program Call Markup Language (PCML), another XML extension that is supported by IBM Toolbox for Java. RFML is neither a subset nor a superset of PCML, but rather a kind of sibling language that adds a few new elements and attributes and omits others.

PCML provides an XML-oriented alternative to using the ProgramCall and ProgramParameter classes. Similarly, RFML provides a user-friendly, easily maintainable alternative to the Record, RecordFormat, and FieldDescription classes.

For more information about RFML, see the following topics:

Requirements
Read about the requirements for using RFML.

RFML example
See how using RFML in your application reduces the amount and sometimes the complexity of code that you must write. The example includes a sample RFML source file.

RecordFormatDocument class
Read about how to use the RecordFormatDocument class with other Toolbox for Java classes to read and write data.

RFML documents and RFML syntax
Learn about RFML documents, called RFML source files, and RFML syntax as defined in the RFML data type definition.

RFML is only one way to use XML with your server. For more information about using XML with iSeries servers, see IBM Toolbox for Java XML extensions and Extensible Markup Language (XML).

Requirements for using RFML

The RFML component has the same workstation Java virtual machine requirements as the rest of the IBM Toolbox for Java.

In addition, in order to parse RFML at run-time, the CLASSPATH for the application must include an XML parser. The XML parser must extend class org.apache.xerces.parsers.SAXParser. For more information, see the following page:

"XML parser and XSLT processor" on page 393

Note: RFML has the same parser requirements as PCML. As with PCML, if you preserialize the RFML file, you do not need to include an XML parser in the application CLASSPATH to run the application.

Related reference

"Workstation requirements for IBM Toolbox for Java" on page 8
Ensure that your workstation meets the following requirements.

Example: Using RFML compared to using IBM Toolbox for Java Record classes

This example illustrates the differences between using RFML and using the IBM Toolbox for Java Record classes.

Using the traditional Record classes, you interweave the data format specifications with the business logic of your application. Adding, changing, or deleting a field means that you must edit and recompile your Java code. However, using RFML isolates the data format specifications into RFML source files that are

entirely separate from the business logic. Accommodating field changes means modifying the RFML file, often without having to change or recompile your Java application.

The example assumes that your application deals with customer records, which you have defined in an RFML source file and named qcustcdt.rfml. The source file represents the fields that compose each customer record.

The listing below shows how a Java application might interpret a customer record using the IBM Toolbox for Java Record, RecordFormat, and FieldDescription classes:

```
// 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"));
```

By comparison, here is how the same record might be interpreted using RFML.

The Java code to interpret the contents of the customer data record using RFML might look like this:

```
    // 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 class

The RecordFormatDocument class enables your Java programs to convert between RFML representations of data and Record and RecordFormat objects for use with other IBM Toolbox for Java components.

RecordFormatDocument class

The RecordFormatDocument class represents an RFML source file, and it provides methods that allow your Java program to perform the following actions:

- Compose RFML source files from Record objects, RecordFormat objects, and byte arrays
- Generate Record objects, RecordFormat objects, and byte arrays that represent the information that the RecordFormatDocument object contains
- Get and set the values of different objects and data types
- Generate XML (RFML) that represents the data that the RecordFormatDocument object contains
- Serialize the RFML source file that the RecordFormatDocument object represents

For more information about the available methods, see the javadoc method summary for the RecordFormatDocument class.

Using the RecordFormatDocument class with other IBM Toolbox for Java classes

Use the RecordFormatDocument class with the following IBM Toolbox for Java classes:

- Record-oriented classes, which include the record-level access file classes (AS400File, SequentialFile, and KeyedFile) that read, manipulate, and write Record objects. This category also includes the LineDataRecordWriter class.
- Byte-oriented classes, which include certain DataQueue, UserSpace, and IFSFile classes that read and write a byte-array of data at a time.

Do not use RecordFormatDocument class with the following IBM Toolbox for Java classes, which read and write data in forms that RecordFormatDocument does not handle:

- The DataArea classes because the read and write methods deal only with String, boolean, and BigDecimal data types.
- IFSTextFileInputStream and IFSTextFileOutputStream because these read and write methods deal only with String.
- JDBC classes because RFML focuses only on data described by the iSeries data description specification (DDS).

Record format documents and RFML syntax

RFML documents, called RFML source files, contain tags that define the specification for a particular data format.

Because RFML is based on PCML, the syntax is familiar to PCML users. Because RFML is an XML extension, RFML source files are easy to read and simple to create. For example, you can create an RFML

source file by using a simple text editor. Also, RFML source files reveal the structure of the data in a way that is easier to understand than in a programming language like Java.

The RFML example Using RFML compared to using IBM Toolbox for Java Record classes includes an example RFML source file.

RFML DTD

The RFML document type definition (DTD) defines valid RFML elements and syntax. To ensure that an XML parser can validate your RFML source file at runtime, declare the RFML DTD in the source file:

```
<!DOCTYPE rfml SYSTEM "rfml.dtd">
```

The RFML DTD resides in the jt400.jar file (com/ibm/as400/data/rfml.dtd).

RFML syntax

The RFML DTD defines tags, each of which has its own attribute tags. You use the RFML tags to declare and define the following elements in your RFML source files:

- The rfml tag begins and ends the RFML source file that describes the data format.
- The struct tag defines a named structure that you can reuse within the RFML source file. The structure contains a data tag for each field in the structure.
- The recordformat tag defines a record format, which contains either data elements or references to structure elements.
- The data tag defines a field within a record format or structure.

In the following example, RFML syntax describes one record format and one structure:

```
<rfml>

    <recordformat>
        <data> </data>
    </recordformat>

    <struct>
        <data> </data>
    </struct>

</rfml>
```

RFML document type definition (DTD):

This is the RFML DTD. Note that the version is 4.0. The RFML DTD resides in the jt400.jar file (com/ibm/as400/data/rfml.dtd).

```
<!--
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;">
```

The RFML data tag:

The data tag can have the following attributes.

Attributes enclosed in brackets, [], indicate that the attribute is optional. If you specify an optional attribute, do not include the brackets in your source. Some attribute values are shown as a list of choices enclosed in braces, {}, with possible choices separated by vertical bars, |. When you specify one of these attributes, do not include the braces in your source and only specify one of the choices shown.

```
<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>
```

The following table lists the data tag attributes. Each entry includes the attribute name, the possible valid values, and a description of the attribute.

Attribute	Value	Description
type=	*char* A character value. A *char* data value is returned as a *java.lang.String*. For more information, see the *char* values for length. *int* An integer value. An *int* data value is returned as a *java.lang.Long*. For more information, see the *int* values for length and precision. *packed* A packed decimal value. A *packed* data value is returned as a *java.math.BigDecimal*. For more information, see the *packed* values for length and precision. *zoned* A zoned decimal value. A *zoned* data value is returned as a *java.math.BigDecimal*. For more information, see the *zoned* values for length and precision. *float* A floating point value. The **length** attribute specifies the number of bytes: either `4` or `8`. A 4-byte integer is returned as a *java.lang.Float*. An 8-byte integer is returned as a *java.lang.Double*. For more information, see the *float* values for length. *byte* A byte value. No conversion is performed on the data. A *byte* data value is returned as an array of *byte* values (*byte[]*). For more information, see the *byte* values for length. *struct* The name of the **<struct>** element. A *struct* allows you to define a structure once and reuse it multiple times within the document. When you use **type=**`"struct"`, it is as if the structure specified appears at this location in the document. A *struct* does not allow for a length value and has no value for precision.	Indicates the type of data being used (character, integer, packed, zoned, floating point, byte, or struct). Values for the length and precision attributes are different for different data types. For more information, see the Values for length and precision.

Attribute	Value	Description
bidistringtype=	*DEFAULT* where DEFAULT is the default string type for non-bidirectional data (LTR). *ST4* where *ST4* is String Type 4. *ST5* where *ST5* is String Type 5. *ST6* where *ST6* is String Type 6. *ST7* where *ST7* is String Type 7. *ST8* where *ST8* is String Type 8. *ST9* where *ST9* is String Type 9. *ST10* where *ST10* is String Type 10. *ST11* where *ST11* is String Type 11.	Specifies the bidirectional string type for **<data>** elements with **type=**"`char`". If this attribute is omitted, string type for this element is implied by the CCSID (whether explicitly specified or the default CCSID of the host environment). String types are defined in the javadoc for the BidiStringType class.
ccsid=	*number* where *number* defines a fixed, never-changing CCSID. *data-name* where *data-name* defines the name that will contain, at runtime, the CCSID of the character data. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=**"`int`". See Resolving Relative Names for more information on how relative names are resolved.	Specifies the host coded character set identifier (CCSID) for character data for the **<data>** element. The **ccsid** attribute can be specified only for **<data>** elements with **type=**"`char`". If this attribute is omitted, character data for this element is assumed to be in the default CCSID of the host environment.
count=	*number* where *number* defines a fixed, never-changing number of elements in a sized array. *data-name* where *data-name* defines the name of a **<data>** element within the RFML document that will contain, at runtime, the number of elements in the array. The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=**"`int`". See Resolving Relative Names for more information on how relative names are resolved.	Specifies that the element is an array and identifies the number of entries in the array. If the `count` attribute is omitted, the element is not defined as an array, although it may be contained within another element that is defined as an array.

Attribute	Value	Description
init=	*string*	Specifies an initial value for the **<data>** element. The initial value specified is used to initialize scalar values. If the element is defined as an array or is contained within a structure defined as an array, the initial value specified is used as an initial value for all entries in the array.
length=	*number* where *number* defines a fixed, never-changing length. *data-name* where *data-name* defines the name of a **<data>** element within the RFML document that will contain, at runtime, the length. A *data-name* can be specified only for **<data>** elements with **type=**"`char`" or **type=**"`byte`". The *data-name* specified can be a fully qualified name or a name that is relative to the current element. In either case, the name must reference a **<data>** element that is defined with **type=**"`int`". See Resolving Relative Names for more information on how relative names are resolved.	Specifies the length of the data element. Usage of this attribute varies depending on the data type. For more information, see the Values for length and precision.
name=	*name*	Specifies the name of the **<data>** element.
precision=	*number*	Specifies the number of bytes of precision for some numeric data types. For more information, see the Values for length and precision.
struct=	*name*	Specifies the name of a **<struct>** element for the **<data>** element. A **struct** attribute can be specified only for **<data>** elements with **type=**"`struct`".

The RFML rfml tag:

The rfml tag can have the following attributes.

Attributes enclosed in brackets, [], indicate that the attribute is optional. If you specify an optional attribute, do not include the brackets in your source.

```
<rfml version="version-string"
    [ ccsid="number" ]>
</rfml>
```

The following table lists the rfml tag attributes. Each entry includes the attribute name, the possible valid values, and a description of the attribute.

Attribute	Value	Description
version=	*version-string* A fixed version of the RFML DTD. For V5R3, `4.0` is the only valid value.	Specifies the version of the RFML DTD, which you can use to verify the correct value.

Attribute	Value	Description
ccsid=	*number* A fixed, never-changing coded character set identifier (CCSID).	Specifies the host CCSID, which applies to all enclosed <**data type**="char"> elements that do not specify a CCSID. For more information, see the RFML **<data>** tag. When you omit this attribute, the default CCSID of the host environment is used.

The RFML recordformat tag:

The recordformat tag can have the following attributes.

Attributes enclosed in brackets, [], indicate that the attribute is optional. If you specify an optional attribute, do not include the brackets in your source.

```
<recordformat name="name"
    [ description="description" ]>
</recordformat>
```

The following table lists the recordformat tag attributes. Each entry includes the attribute name, the possible valid values, and a description of the attribute.

Attribute	Value	Description
name=	*name*	Specifies the name of the recordformat.
description=	*description*	Specifies the description of the recordformat.

The RFML struct tag:

The struct tag can have the following attributes.

Attributes enclosed in brackets, [], indicate that the attribute is optional. If you specify an optional attribute, do not include the brackets in your source.

```
<struct name="name">
</struct>
```

The following table lists the struct tag attributes. Each entry includes the attribute name, the possible valid values, and a description of the attribute.

Attribute	Value	Description
name=	*name*	Specifies the name of the **<struct>** element.

XML parser and XSLT processor

Some IBM Toolbox for Java packages or functions require that, at run-time, you have an Extensible Markup Language (XML) parser or Extensible Stylesheet Language Transformations (XSLT) processor in your CLASSPATH environment variable.

Refer to the following information to determine which parser and processor you want to use.

For more information about which IBM Toolbox for Java packages and functions require an XML parser or XSLT processor, see the following page:

"Jar files" on page 11

XML parser

If the package or function requires an XML parser, you must include an XML parser in the CLASSPATH at run-time. The XML parser must meet the following requirements:

- Be JAXP-compliant
- Extend class org.apache.xerces.parsers.SAXParser
- Support full schema validation

Note: The parser only needs to support full schema validation if use intend to use XPCML. If you are only using PCML, full schema validation is not necessary.

The Java 2 Software Developer Kit (J2SDK), version 1.4, includes a suitable XML parser. If you use a previous version of J2SDK, use any of the following methods to access a suitable XML parser:

- Use x4j400.jar (an IBM version of the Xerces XML parser from Apache)
- Download the Xerces XML parser from the Apache Web site 
- Use any compatible XML parser in the /QIBM/ProdData/OS400/xml/lib directory on your iSeries server

Note: Consider using the latest versions of the parsers that reside in /QIBM/ProdData/OS400/xml/lib. For example, xmlapis11.jar and xerces411.jar are both fully validating parsers.

You can use these parsers on your server or copy them to a workstation.

Note: Any XML parser that meets the requirements for running XPCML can run PCML and RFML. Keep in mind that XPCML does not support serialization.

XSLT processor

If the package or function requires an XSLT processor, you must include an XSLT processor in the CLASSPATH at run-time. The XSLT processor must meet the following requirements:

- Be JAXP-compliant
- Contain the class javax.xml.transform.Transformer

The Java 2 Software Developer Kit (J2SDK), version 1.4, includes a suitable XSLT processor. If you use a previous version of J2SDK, use any of the following methods to access a suitable XSLT processor:

- Use xslparser.jar (an IBM version of the Xalan XSLT processor from Apache)
- Download the Xalan XSLT processor from the Apache Web site 
- Use any compatible XSLT processor in the /QIBM/ProdData/OS400/xml/lib directory on your iSeries server

You can use these processors on your server or copy them to a workstation.

Extensible Program Call Markup Language

Extensible Program Call Markup Language (XPCML) enhances the functionality and usability of the Program Call Markup Language (PCML) by offering support for XML schemas. XPCML does not support serialization, so unlike PCML, you cannot serialize an XPCML document.

However, XPCML offers several valuable enhancements when compared to PCML:

- Specify and pass values for program parameters
- Retrieve the results of a program call to your iSeries server in XPCML
- Transform an existing PCML document into the equivalent XPCML document

- Extend and customize the XPCML schema to define new simple and complex elements and attributes

For more information about XPCML, see the following pages:

Advantages of XPCML over PCML
Find out more about valuable XPCML enhancements compared to PCML.

Requirements
Read about the software requirements for using XPCML.

XPCML schema and syntax
See how the XPCML schema defines XPCML syntax and available server data types. Find out how to extend and customize the schema to meet your specific programming needs.

Using XPCML
Learn how to take advantage of XPCML enhancements, including passing different kinds of parameters to your server program and retrieving returned parameter data. Find out how to condense XPCML code, which makes the code easier to use and read, and how to use XPCML with your current PCML-enabled applications.

XPCML is only one way to use XML with your server. For more information about using XML, see the following pages:

Extensible Markup Language components

XML Toolkit

W3C Architecture domain: XML Schema 

Advantages of XPCML over PCML

Extensible Program Call Markup Language (XPCML) offers several valuable enhancements over PCML.

- Specify and pass values for program parameters
- Retrieve the results of a program call to your iSeries server in XPCML
- Transform an existing PCML document into the equivalent XPCML document
- Extend and customize the XPCML schema to define new simple and complex elements and attributes

Specify and pass values for program parameters

XPCML uses an XML schema to define program parameter types; PCML uses a data type definition (DTD). At parse time, the XML parser validates data values entered as parameters against the appropriate parameters as defined in the schema. Data types exist for parameters of many types: strings, integers, longs, and so on. This ability to specify and pass values for program parameters is a significant improvement over PCML. In PCML, you can verify values for parameters only after parsing the PCML document. Additionally, verifying parameter values in PCML often requires coding your application to perform the validation.

Retrieve results of a program call in XPCML

XPCML also provides the capability to retrieve program call results as XPCML. In PCML, you obtain program call results by calling one of the getValue methods of the ProgramCallDocument class after you make the call to the program. In XPCML, you can use the getValue methods, but you can also have your XPCML call a generateXPCML method, which returns the results of a program call as XPCML.

Transform existing PCML documents into XPCML

A new method of the ProgramCallDocument class, transformPCMLToXPCML, enables you to transform existing PCML documents to equivalent XPCML documents. This allows you to take advantage of new XPCML function without writing XPCML source for your existing iSeries program call documents.

Extend and customize the XPCML schema

XPCML is extensible which means you can define new parameter types that extend those specified by the XPCML schema. Condensing XPCML extends the XPCML schema to create new data type definitions that simplify and improve the readability and usability of your XPCML documents.

Requirements for using XPCML

The Extensible Program Call Markup Language (XPCML) has the same workstation Java virtual machine requirements as the rest of the IBM Toolbox for Java.

For more information, see the following page:

"Workstation requirements for running IBM Toolbox for Java applications" on page 8

In addition, using XPCML includes requirements for the XML schema file, XML parser, and Extensible Stylesheet Language Transformation (XSLT) processor:

XML schema file

Your XPCML documents must know the location of the file that contains the schema. The default schema for IBM Toolbox for Java is xpcml.xsd, which resides within the jt400.jar file.

To use the default schema, extract xpcml.xsd from jt400.jar, then place the file in a suitable location. The following procedure shows one way that you can extract the .xsd file on a workstation.

Extract the xpcml.xsd schema file

- Start a command line session in the directory that contains jt400.jar
- Use the following command to extract the .xsd file:

```
jar xvf  jt400.jar   com/ibm/as400/data/xpcml.xsd
```

Note: If you do not run the previous command from the directory that contains jt400.jar, you can specify a fully qualified path to jt400.jar.

You can place the default schema file (or any schema file) in any directory. The only requirement is that you specify the location of the schema file by using the xsi:noNamespaceSchemaLocation attribute in the <xpcml> tag.

You can specify the location of the schema as a file path or as a URL.

Note: Although the following examples use xpcml.xsd as the schema file, you can specify any schema that extends xpcml.xsd.

- To specify the same directory as the XPCML file, use xsi:noNamespaceSchemaLocation='xpcml.xsd'
- To specify a fully qualified path: xsi:noNamespaceSchemaLocation='c:\myDir\xpcml.xsd'
- To specify a URL: xsi:noNamespaceSchemaLocation='http://myServer/xpcml.xsd'

To see an HTML version of the xpcml.xsd file, see the following page:

"Schema xpcml.xsd file" on page 398

XML parser and XSLT processor

At run-time, you must include an XML parser and an XSLT processor in your CLASSPATH environment variable. For more information, see the following page:

"XML parser and XSLT processor" on page 393

XPCML schema and syntax

XPCML documents, called XPCML source files, contain tags and data that fully define calls to programs on your iSeries server.

Because XPCML uses XML schemas instead of a document type definition (DTD), you can use XPCML in ways that you cannot use PCML:

- Pass values for input parameters to your program as XML elements
- Receive values for output parameters from your program as XML elements
- Have the XML parser automatically validate the values passed to your program
- Extend the schema to define new simple and complex elements

For more information about the XPCML schema and syntax, see the following pages:

Comparison of XPCML source to PCML source
Examine examples that compare XPCML source and PCML source. The examples illustrate how XPCML provides more functionality and makes the source data easier to read and write.

XPCML schema
Examine the XPCML schema file and learn more about using and extending the XPCML schema.

XPCML syntax
Review a list of XPCML syntax elements that the schema uses to define the XPCML elements.

XPCML tag attributes
Description of the different attributes for each element XPCML schema defines.

Comparison of XPCML source to PCML source:

XPCML differs from PCML in several ways, but one major difference is that XPCML allows you to specify the values of input parameters within the XPCML source file.

PCML allows you to use the init attribute of the <data> tag to specify the initial value for a data element in the PCML source. However, using PCML to specify values has the following limitations:

- You cannot use the init attribute to set array values
- Validation of the init value occurs only after parsing the PCML document

To specify array values in PCML, you must first read in and parse the PCML document, then perform a series of calls to ProgramCallDocument.setValue().

Using XPCML makes it easier to specify values of single elements and arrays:

- Specify values for both scalar and array elements in the XPCML source file
- Validate the specified array values at parse time

The following simple comparisons indicate ways in which XPCML differs from PCML. Each example defines a program call for an iSeries server program.

Example: Calling an iSeries server program

The following examples call an iSeries server program called prog1.

XPCML source code

```
<?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 source code

```
<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>
```

Example: Calling an iSeries server program using an array of string parameters

The following examples call an iSeries server program called prog2 and define parm1 as an array of string parameters. Note the functionality of XPCML:

- Initializes the value of each element in the array
- Specifies the input values as element content that a fully validating XML parser can verify

You can take advantage of this XPCML functionality without writing any Java code.

PCML cannot match the XPCML performance. PCML cannot initialize the value of each element in the array. PCML cannot validate the init values at parse time. To match the functionality of XPCML, you would have to read in and parse the PCML document, then code your Java application to set the value for each array element. You would also have to write code to validate the parameters.

XPCML source code

```
<?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 source code

```
<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>
```

Schema xpcml.xsd file:

For more information about using the xpcml.xsd file, see Requirements for using XPCML.

Note: Read the Code example disclaimer for important legal information.

To make it easier to display and print, some lines of this HTML version of xpcml.xsd wrap to a second line. The same lines in the source xsd file appear on a single line.

```
<?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) 1997-2003 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 iSeries program is on same server -->
    <!-- and is thread-safe, set to true to call the iSeries 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 i5/OS on which this element exists. -->
    <xs:attribute name="minvrm" type="string10" />
    <!-- The highest version of i5/OS 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 i5/OS 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 i5/OS 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 iSeries 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 syntax:

The XPCML schema defines several element tags, and each element tag contains attribute tags.

The following table lists the different elements that you can declare and define in your XPCML source files. Each entry in the first column links to the appropriate section of the XPCML schema.

XPCML tag	Description	Equivalent PCML tag
doubleParm	Defines a double parameter	data (type=float, length=8)
arrayOfDoubleParm	Defines a parameter that is an array of doubles	
floatParm	Defines a float parameter	data (type=float, length=4)
arrayOfFloatParm	Defines a parameter that is an array of floats	
hexBinaryParm	Defines a byte parameter represented in hex	byte (rough equivalent, represented in hex)
arrayOfHexBinaryParm	Defines a parameter that is an array of hexBinaries	
intParm	Defines an integer parameter	data (type=int, length=4)
arrayOfIntParm	Defines a parameter that is an array of integers	
longParm	Defines a long parameter	data (type=int, length=8)
arrayOfLongParm	Defines a parameter that is an array of longs	
packedDecimalParm	Defines a packed decimal parameter	data (type=packed)
arrayOfPackedDecimalParm	Defines a parameter that is an array of packed decimals	
parameterList	Signals that the enclosing tag represents all of the parameter definitions for the program	
program	Begins and ends the XML that describes one program call	program
shortParm	Defines a short parameter	data (type int, length 2)
arrayOfShortParm	Defines a parameter that is an array of shorts	
stringParm	Defines a string parameter	
arrayOfStringParm	Defines a parameter that is an array of strings	
struct	Defines a named structure that you can specify as an argument to a program or as a field within another named structure	struct
arrayOfStruct	Defines an array of structs	
structParm	Represents a reference to a struct tag found elsewhere in the XPCML document that you want to included at a specific location in the document	data (type=struct)
arrayOfStructParm	Defines a parameter that is an array of struct parameters	
unsignedIntParm	Defines an unsigned integer parameter	data (type=int, length=4, precision=32)
arrayOfUnsignedIntParm	Defines a parameter that is an array of unsigned integers	

XPCML tag	Description	Equivalent PCML tag
unsignedShortParm	Defines an unsigned short parameter	data (type=int, length=2, precision=16)
arrayOfUnsignedShortParm	Defines a parameter that is an array of unsigned shorts	
xpcml	Begins and ends the XPCML source file that describes the program call format	
zonedDecimalParm	Defines a zoned decimal parameter	data (type zoned)
arrayOfZonedDecimalParm	Defines a parameter that is an array of zoned decimals	

XPCML tag attributes:

The XPCML schema defines several element tags, and each element tag contains attribute tags. The following table lists and describes the different attributes for each element.

For more specific and detailed information about XPCML tags and their attributes, see XPCML schema.

XPCML tag	Attribute	Description
hexBinaryParm	fill last 2 columns with data and decide format	
arrayOfHexBinaryParm		
doubleParm	Defines a double parameter	float (length 8)
arrayOfDoubleParm	Defines a parameter that is an array of doubles	
floatParm	Defines a float parameter	data (type float, length 4)
arrayOfFloatParm	Defines a parameter that is an array of floats	
intParm	Defines an integer parameter	data (type int, length 4)
arrayOfIntParm	Defines a parameter that is an array of integers	
longParm	Defines a long parameter	data (type int, length 8)
arrayOfLongParm	Defines a parameter that is an array of longs	
packedDecimalParm	Defines a packed decimal parameter	data (type packed)
arrayOfPackedDecimalParm	Defines a parameter that is an array of packed decimals	
parameterList	Signals that the enclosing tag represents all of the parameter definitions for the program	
program	Begins and ends the XML that describes one program call	
shortParm	Defines a short parameter	data (type int, length 2)
arrayOfShortParm	Defines a parameter that is an array of shorts	
stringParm	Defines a string parameter	

XPCML tag	Attribute	Description
arrayOfStringParm	Defines a parameter that is an array of strings	
struct	Defines a named structure that you can specify as an argument to a program or as a field within another named structure	
arrayOfStruct	Defines an array of structs	
structParm	Represents a reference to a struct tag found elsewhere in the XPCML document that you want to included at a specific location in the document	data (type struct)
arrayOfStructParm	Defines a parameter that is an array of struct parms	
unsignedIntParm	Defines an unsigned integer parameter	data (type int, length 4, precision 32)
arrayOfUnsignedIntParm	Defines a parameter that is an array of unsigned integers	
unsignedShortParm	Defines an unsigned short parameter	data (type int, length 2, precision 16)
arrayOfUnsignedShortParm	Defines a parameter that is an array of unsigned shorts	
xpcml	Begins and ends the XPCML source file that describes the program call format	
zonedDecimalParm	Defines a zoned decimal parameter	data (type zoned)
arrayOfZonedDecimalParm	Defines a parameter that is an array of zoned decimals	

Using XPCML

Using XPCML is similar to using PCML. The following steps serve as a general outline for the actions that you need to perform to use XPCML.

1. Use XPCML to describe the specification for the program call
2. Create a ProgramCallDocument object
3. Use ProgramCallDocument.callProgram() to run the program

Despite the similarities to using PCML, using XPCML offers enhanced functionality:

- Have the parser automatically validate parameter values
- Specify and pass values for program parameters
- Retrieve the results of a program call to your iSeries server in XPCML
- Transform an existing PCML document into the equivalent XPCML document
- Extend the XPCML schema to define new simple and complex elements and attributes

For example, IBM Toolbox for Java enables you to extend the XPCML schema to create new parameter and data types. You can use this XPCML capability to condense your XPCML source files, which makes the files easier to read and code easier to use.

For more information about using XPCML, see the following pages:

Converting PCML to XPCML
Learn how to transform existing PCML documents to equivalent XPCML documents.

Using XPCML to call a program on your iSeries server
See how to create a ProgramCallDocument object that uses XPCML to call a program on your server.

Obtaining program call results as XPCML
Find out how to retrieve the results of your call to a server program as XPCML.

Passing in parameter values as XPCML
Learn about setting values for program parameters in XPCML and passing in those values. Examine different techniques for defining constant parameter values and data arrays.

Using condensed XPCML
See how condensing an XPCML document makes it easier to read and use. Find out about using condensed XPCML to create a ProgramCallDocument object and obtaining program call results as condensed XPCML.

Identifying parse errors in XPCML
Find out how to identify and log useful warnings and non-fatal parser errors that sometimes occur when parsing an XPCML document.

Converting existing PCML to XPCML:

The ProgramCallDocument class contains the transformPCMLToXPCML method that enables you to transform existing PCML documents to equivalent XPCML documents.

XPCML has comparable definitions for all of the elements and attributes that you can define in PCML. Using transformPCMLToXPCML() converts the PCML representation of the elements and attributes to the equivalent XPCML.

Note that in some cases, equivalent XPCML attributes have a different name than in PCML. For example, the attribute ″usage″ in PCML is the attribute ″passDirection″ in XPCML. For more information about how to use XPCML compared to PCML, see XPCML schema and syntax.

The method takes the existing PCML document, which you pass to it as an InputStream object, and generates the equivalent XPCML as an OutputStream object. Because transformPCMLToXPCML() is a static method, you can call it without first creating a ProgramCallDocument object.

Example: Converting a PCML document to an XPCML document

The following example shows how to convert a PCML document (named myPCML.pcml) to an XPCML document (named myXPCML.xpcml).

Note: You must specify .xpcml as the file extension for XPCML files. Using .xpcml as the file extension ensures that the ProgramCallDocument class recognizes the file as XPCML. If you do not specify an extension, ProgramCallDocument assumes that the file is PCML.

PCML document 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>
```

Java code to convert myPCML.pcml to myPCML.xpcml

```
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();
}
```

Resulting XPCML document 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>
```

For more information about transformPCMLToXPCML() and the ProgramCallDocument class, see the following page:

ProgramCallDocument javadoc information

Using XPCML to call a program on your iSeries server:

After you create your XPCML file, you need to create a ProgramCallDocument object that can use the XPCML specifications and data values to call a program on your iSeries server.

Create an XPCML ProgramCallDocument by passing in the name of your XPCML file on the ProgramCallDocument constructor. Creating an XPCML ProgramCallDocument in this way first parses and validates your XPCML document, then creates the ProgramCallDocument object.

In order to parse and validate the XPCML document, make sure that your CLASSPATH includes a fully validating XML parser. For more information about requirements to run XPCML, see the following page:

"XML parser and XSLT processor" on page 393

The following example shows how to create a ProgramCallDocument object for the XPCML file, myXPCML.xpcml.

```
system = new AS400();
// Create a ProgramCallDocument into which to parse the file.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "myXPCML.xpcml");
```

The only difference between creating an XPCML ProgramCallDocument and a PCML ProgramCallDocument is that you pass the constructor an XPCML document instead of a PCML document.

Note: You must specify .xpcml as the file extension for XPCML files. Using .xpcml as the file extension ensures that the ProgramCallDocument class recognizes the file as XPCML. If you do not specify an extension, ProgramCallDocument assumes that the file is PCML.

Using XPCML to call a program on your iSeries server

After you create the ProgramCallDocument object, use any of the methods of the ProgramCallDocument class to work with your XPCML document. For example, call an iSeries program by using ProgramCallDocument.callProgram() or change the value for an XPCML input parameter before calling the server program by using the appropriate ProgramCallDocument.setValue method.

The following example shows how to create a ProgramCallDocument object for an XPCML file (named myXPCML.xpcml). After creating the ProgramCallDocument object, the example calls a program (PROG1) that is specified in the XPCML document. In this case, the only difference between using XPCML and PCML is that the example passes an XPCML file to the ProgramCallDocument constructor.

After your application reads and parses an XPCML document, the XPCML document functions just like a PCML document. At this point, XPCML can use any of the existing methods that PCML uses.

```
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");
```

Obtaining program call results as XPCML:

After you call a server program, you can use ProgramCallDocument.getValue methods to retrieve the Java objects that represent program parameter values.

Additionally, the following generateXPCML methods enable ProgramCallDocument to return the results of a program call as XPCML:

- generateXPCML(String fileName): Generates results in XPCML for the entire XPCML source file that you used to construct the ProgramCallDocument object. Stores the XPCML in a file with the specified file name.
- generateXPCML(String pgmName, String fileName): Generates results in XPCML for only the specified program and its parameters. Stores the XPCML in a file with the specified file name.
- generateXPCML(java.io.OutputStream outputStream): Generates results in XPCML for the entire XPCML source file. Stores the XPCML in the specified OutputStream object.
- generateXPCML(String pgmName, java.io.OutputStream outputStream): Generates results in XPCML for only the specified program and its parameters. Stores the XPCML in the specified OutputStream object.

For more information about the ProgramCallDocument class, see the following page:

ProgramCallDocument javadoc information

The following example shows how you can construct an XPCML ProgramCallDocument, call an iSeries program, and retrieve the results of the program call as XPCML.

"Example: Retrieving program call results as XPCML" on page 715

Passing in parameter values as XPCML:

You can set the values for program parameters in the XPCML source file and pass in the parameter values as XPCML.

When the ProgramCallDocument object reads in and parses the XPCML document, it automatically calls the appropriate setValue method for each parameter specified in the XPCML.

Using XPCML to pass in parameter values relieves you from writing Java code that sets the values of complicated structures and arrays.

The following examples show different ways to construct arrays and pass in parameter values as XPCML:

"Example: Passing in parameter values as XPCML" on page 718

"Examples: Passing in arrays of parameter values as XPCML" on page 719

Using condensed XPCML:

Because XPCML is extensible, you can define new parameter types that extend those specified by the XPCML schema. Condensing XPCML extends the XPCML schema to create new data type definitions that simplify and improve the readability and usability of your XPCML documents.

The following discussion assumes that you understand the XPCML schema. For more information about the XPCML schema, see the following page:

"XPCML schema and syntax" on page 397

To condense existing XPCML source, you use the ProgramCallDocument.condenseXPCML method, which generates the following:

- An extended schema that contains new type definitions for each parameter in the existing XPCML source
- New XPCML source that uses the type definitions provided in the extended schema

For more information about condensing your XPCML, see the following pages:

"Using condensed XPCML"

"Example: Using condensed XPCML to create a ProgramCallDocument object" on page 724

"Example: Obtaining program call results as condensed XPCML" on page 724

Condensing existing XPCML documents:

Condensing existing XPCML documents results in more readable and usable XPCML source. To create condensed XPCML, use the ProgramCallDocument.condenseXPCML method.

To call condenseXPCML(), provide the following parameters to the method:

- An input stream that represents the existing XPCML
- An ouput stream that repesents the condensed XPCML
- An output stream that represents the new, extended schema
- A name for the new schema in the appropriate format (for example, mySchema.xsd)

For more information about condenseXPCML() and the ProgramCallDocument class, see the following page:

ProgramCallDocument javadoc information

ProgramCallDocument.condenseXPCML() is a static method, which means that you do not have to instantiate a ProgramCallDocument object in order to call the method.

Examples

The following examples illustrate how to condense an existing XPCML document.

The first example is simple and includes original XPCML source, the resulting condensed XPCML, and the extended schema. The second example is longer and more complex, so it includes the Java code that calls condenseXPCML() and only a few of the newly generated type definitions in the extended schema:

"Example: Condensing an existing XPCML document" on page 721

"Example: Condensing an existing XPCML document" on page 721

Identifying parse errors in XPCML:

When validating XPCML schema documents, a fully validating XML parser may generate warnings, non-fatal parse errors, and fatal parse errors.

Warnings and non-fatal parse errors do not cause the parse to fail. You might want to examine warnings and non-fatal errors to help you determine problems with your XPCML source. Fatal parse errors cause the parse to terminate with an exception.

To display warnings and non-fatal parser errors when parsing an XPCML document, turn tracing on in your application and set the trace category to PCML.

Example

A fully validating XML parser generates an error for any numeric parameter type that does not have a value. The following example shows sample XPCML source and the resulting non-fatal parse error:

XPCML source

```
<program name="prog1"/>
   <parameterList>
      <intParm name="parm1"/>
   </parameterList>
</program>
```

Resulting error

```
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.
```

To prevent logging this kind of error, add the nil=true attribute to the intParm element. The nil=true atttribute signals to the parser that you have deliberately left the element empty.. Here is the previous XPCML source with the nil=true atttribute added:

```
<program name="prog1"/>
   <parameterList>
      <intParm xsi:nil="true" name="parm1"/>
   </parameterList>
</program>
```

Frequently asked questions (FAQ)

IBM Toolbox for Java frequently asked questions (FAQs) provide answers to questions about optimizing your IBM Toolbox for Java performance, performing troubleshooting, using JDBC, and more.

- IBM Toolbox for Java FAQ : Find answers to many types of questions, including improving performance, using i5/OS, performing troubleshooting, and more.
- IBM Toolbox for Java JDBC FAQ : Find answers to questions about using JDBC with IBM Toolbox for Java

Tips for programming

This section features a variety of tips that can help you use IBM Toolbox for Java.

Shutting down your Java program

To ensure that your program shuts down properly, issue System.exit(0) as the last instruction before your Java program ends.

Note: Avoid using System.exit(0) with servlets because doing so shuts down the entire Java virtual machine.

IBM Toolbox for Java connects to the server with user threads. Because of this, a failure to issue System.exit(0) may keep your Java program from properly shutting down.

Using System.exit(0) is not a requirement, but a precaution. There are times that you must use this command to exit a Java program and it is not problematic to use System.exit(0) when it is not necessary.

Integrated file system path names for server objects

Your Java program must use integrated file system names to refer to server objects, such as programs, libraries, commands, or spooled files. The integrated file system name is the name of a server object as it might be accessed in the library file system of the integrated file system on the iSeries server.

The path name may consist of the following components:

Path name component	Description
library	The library in which the object resides. The library is a **required** portion of an integrated file system path name. The library name must be 10 or fewer characters and be followed by **.lib**.
object	The name of the object that the integrated file system path name represents. The object is a **required** portion of an integrated file system path name. The object name must be 10 or fewer characters and be followed by **.type**, where **type** is the type of the object. Types can be found by prompting for the OBJTYPE parameter on control language (CL) commands, such as the Work with Objects (WRKOBJ) command.
type	The type of the object. The type of the object must be specified when specifying the **object**. (See **object** above.) The type name must be 6 or fewer characters.
member	The name of the member that this integrated file system path name represents. The member is an **optional** portion of an integrated file system path name. It can be specified only when the **object type** is **FILE**. The member name must be 10 or fewer characters and followed by **.mbr**.

Follow these conditions when determining and specifying the integrated file system name:

- The forward slash (/) is the path separator character.
- The root-level directory, called QSYS.LIB, contains the server library structure.
- Objects that reside in the server library QSYS have the following format:

  ```
  /QSYS.LIB/object.type
  ```

- Objects that reside in other libraries have the following format:

  ```
  /QSYS.LIB/library.LIB/object.type
  ```

- The object type extension is the server abbreviation used for that type of object.

To see a list of these types, enter a CL command that has object type as a parameter and press **F4** (Prompt) for the type. For example, the Work with Objects (WRKOBJ) command has an object type parameter.

The following table is a list of some commonly used object types and the abbreviation for each type:

Object type	Abbreviation
command	.CMD
data queue	.DTAQ
file	.FILE
font resource	.FNTRSC
form definition	.FORMDF
library	.LIB
member	.MBR
overlay	.OVL
page definition	.PAGDFN
page segment	.PAGSET
program	.PGM
output queue	.OUTQ
spooled file	.SPLF

Use the following descriptions to help you determine how to specify integrated file system path names:

Integrated file system name	Description
/QSYS.LIB/MY_LIB.LIB/MY_PROG.PGM	Program MY_PROG in library MY_LIB on the server
/QSYS.LIB/MY_LIB.LIB/MY_QUEUE.DTAQ	Data queue MY_QUEUE in library MY_LIB on the server
/QSYS.LIB/YEAR1998.LIB/MONTH.FILE/JULY.MBR	Member JULY in file MONTH in library YEAR1998 on the server

Integrated file system special values

Various IBM Toolbox for Java classes recognize special values in integrated file system path names. The traditional format for these special values (as used on an iSeries command line) begins with an asterisk (***ALL**). However, in a Java program that uses IBM Toolbox for Java classes, the format for these special values begins and ends with percent signs (**%ALL%**).

Note: In the integrated file system, an asterisk is a wildcard character.

The following table shows which of these special values the IBM Toolbox for Java classes recognize for particular path name components. The table also shows how the traditional format for these special values differ from the format used in IBM Toolbox for Java classes.

Path name component	Traditional format	IBM Toolbox for Java format
Library name	*ALL	%ALL%
	*ALLUSR	%ALLUSR%
	*CURLIB	%CURLIB%
	*LIBL	%LIBL%
	*USRLIBL	%USRLIBL%
Object name	*ALL	%ALL%

Path name component	Traditional format	IBM Toolbox for Java format
Member name	*ALL	%ALL%
	*FILE	%FILE%
	*FIRST	%FIRST%
	*LAST	%LAST%

See the QSYSObjectPathName class for information about building and parsing integrated file system names.

For more information about integrated file system concepts, see Integrated file system concepts.

Managing connections

It is important to be able to create, start, and end connections to your server. The following discussion explains concepts central to managing connections to your server and also offers some code examples.

To connect to an iSeries server, your Java program must create an AS400 object. The AS400 object contains up to one socket connection for each iSeries server type. A service corresponds to a job on the server and is the interface to the data on the server.

Note: When you create Enterprise JavaBeans (EJB), comply with the EJB specification that does not allow threads during your connection. Although turning off IBM Toolbox for Java thread support may slow your application, it is required to comply with the EJB specification.

Every connection to each server has its own job on the iSeries. A different server supports each of the following:

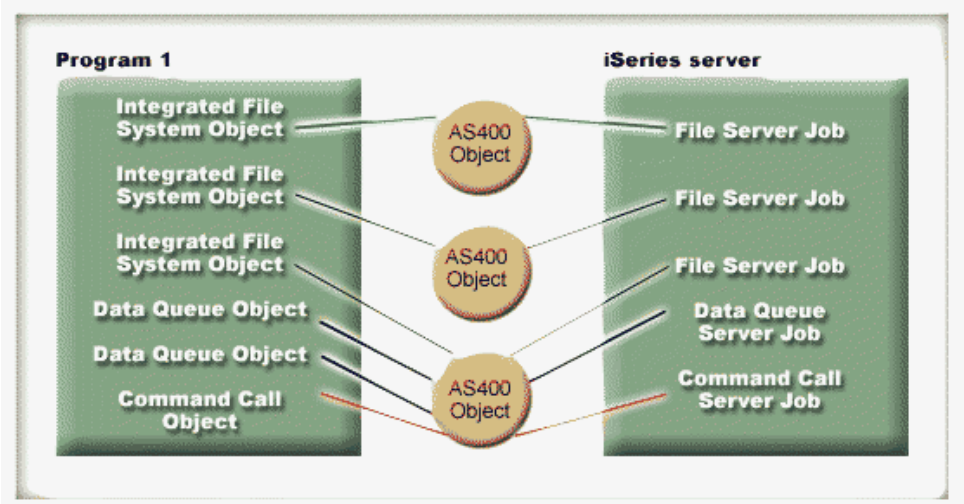
- JDBC
- Program call and command call
- Integrated file system
- Print
- Data queue
- Record-level access

Note:

- The print classes use one socket connection per AS400 object if the application does not try to do two things that require the network print server at the same time.
- A print class creates additional socket connections to the network print server if needed. The extra conversations are disconnected if they are not used for 5 minutes.

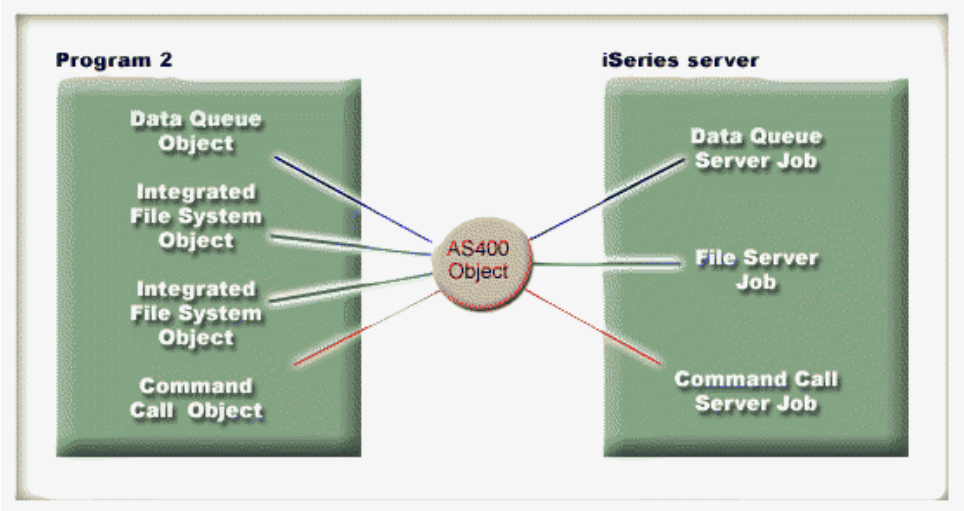
The Java program can control the number of connections to iSeries. To optimize communications performance, a Java program can create multiple AS400 objects for the same server as shown in Figure 1. This creates multiple socket connections to the iSeries server.

Figure 1: Java program creating multiple AS400 objects and socket connections for the same iSeries server



To conserve server resources, create only one AS400 object as shown in Figure 2. This approach reduces the number of connections, which reduces the amount of resource used on the server.

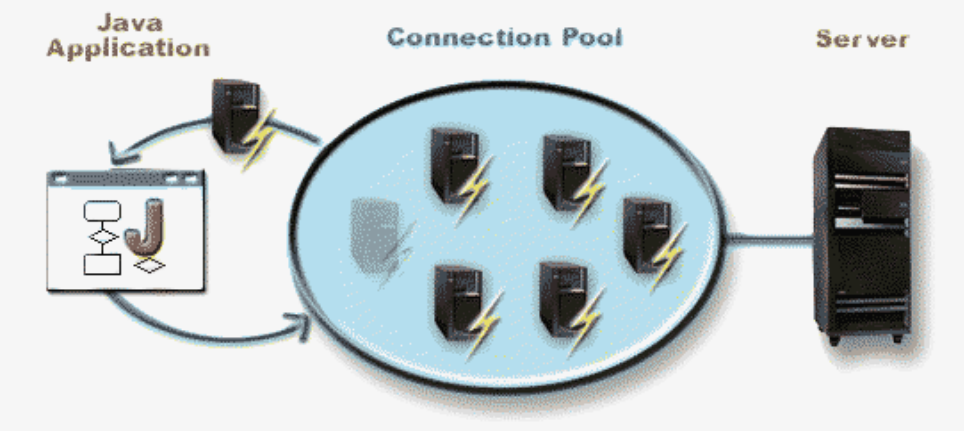
Figure 2: Java program creating a single AS400 object and socket connection for the same iSeries server



Note: Although creating more connections increases the amount of resource used on the server, creating more connections can have a benefit. Having more connections enables your Java program to process things in parallel, which can give better throughput (transactions-per-second) and speed up your application.

You can also choose to use a connection pool to manage connections as shown in Figure 3. This approach reduces the amount of time it takes to connect to iSeries by reusing a connection previously established for the user.

Figure 3: Java program getting a connection from an AS400ConnectionPool to an iSeries server



The following examples show how to create and use AS400 objects:

Example 1: In the following example, two CommandCall objects are created that send commands to the same server. Because the CommandCall objects use the same AS400 object, only one connection to the server is created.

```
        // 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();
```

Example 2: In the following example, two CommandCall objects are created that send commands to the same iSeries server. Because the CommandCall objects use different AS400 objects, two connections to the server are created.

```
        // 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();
```

Example 3: In the following example, a CommandCall object and an IFSFileInputStream object are created using the same AS400 object. Because the CommandCall object and the IFSFileInput Stream object use different services on the iSeries server, two connections are created.

```
        // 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();
```

Example 4: In the following example, an AS400ConnectionPool is used to get an iSeries connection. This example (like Example 3 above) does not specify a service, so the connection to the command service is made when the command is run.

```
        // 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);
```

Example 5: The following example uses AS400ConnectionPool to connect to a particular service when requesting the connection from the pool. This eliminates the time required to connect to the service when the command is run (see Example 4 above). If the connection is returned to the pool, the next call to get a connection can return the same connection object. This means that no extra connection time is needed, either on creation or use.

```
        // 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);
```

Starting and ending connections

The Java program can control when a connection is started and ended. By default, a connection is started when information is needed from the server. You can control exactly when the connection is made by calling the connectService() method on the AS400 object to preconnect to the server.

Using an AS400ConnectionPool, you can create a connection preconnected to a service without calling the connectService() method, as in Example 5 above.

The following examples show Java programs connecting to and disconnecting from iSeries.

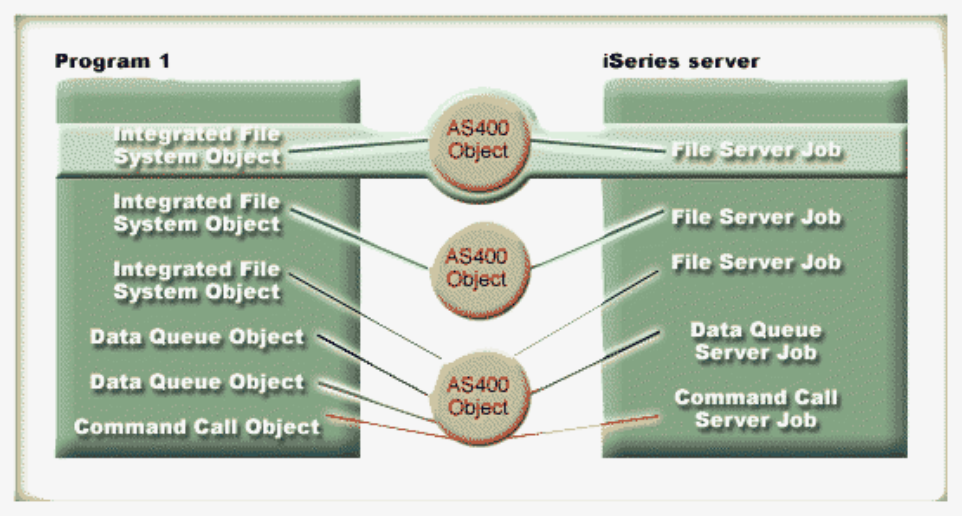
Example 1: This example shows how to preconnect to iSeries:

```
        // 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);
```

Example 2: Once a connection is started, the Java program is responsible for disconnecting, which is done either implicitly by the AS400 object, or explicitly by the Java program. A Java program disconnects by calling the disconnectService() method on the AS400 object. To improve performance, the Java program must disconnect only when the program is finished with a service. If the Java program disconnects before it is finished with a service, the AS400 object reconnects, if it is possible to reconnect, when data is needed from the service.

Figure 4 shows how disconnecting the connection for the first integrated file system object connection ends only that single instance of the AS400 object connection, not all of the integrated file system object connections.

Figure 4: Single object using its own service for an instance of an AS400 object is disconnected

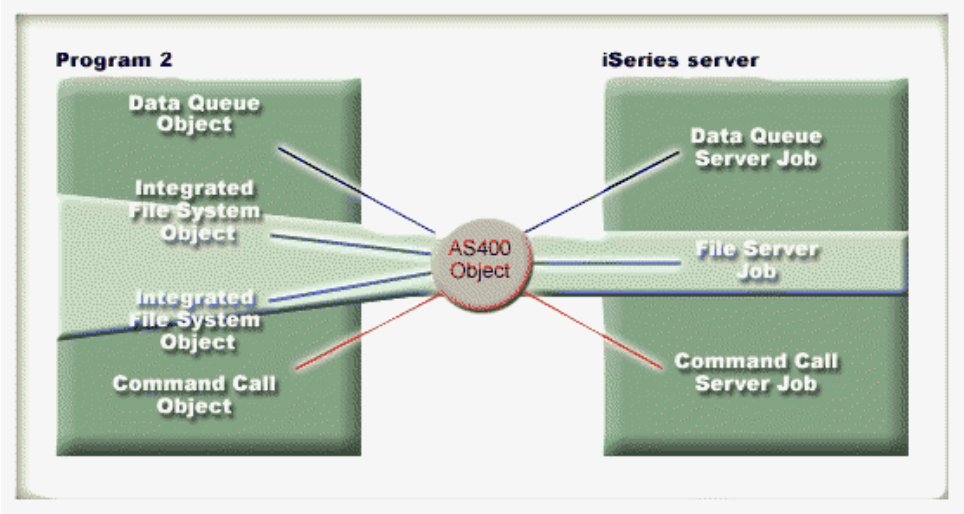


This example shows how the Java program disconnects a connection:

```
        // 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);
```

Example 3: Multiple objects that use the same service and share the same AS400 object share a connection. Disconnecting ends the connection for all objects that are using the same service for each instance of an AS400 object as is shown in Figure 5.

Figure 5: All objects using the same service for an instance of an AS400 object are disconnected



For example, two CommandCall objects use the same AS400 object. When disconnectService() is called, the connection is ended for both CommandCall objects. When the run() method for the second CommandCall object is called, the AS400 object must reconnect to the service:

```
        // 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);
```

Example 4: Not all IBM Toolbox for Java classes automatically reconnect. Some method calls in the integrated file system classes do not reconnect because the file may have changed. While the file was disconnected, some other process may have deleted the file or changed its contents. In the following example, two file objects use the same AS400 object. When disconnectService() is called, the connection is ended for both file objects. The read() for the second IFSFileInputStream object fails because it no longer has a connection to the server.

```
        // 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);
```

Long description of Figure 1: Java program creating multiple AS400 objects and socket connections for the same iSeries server (rzahh549.gif)

found in IBM Toolbox for Java: Managing connections

This figure illustrates how a Java program can create multiple AS400 objects, each of which has a separate socket connection to the same iSeries server. Creating multiple connections increases the amount of resource used on the system.

Description

The figure is composed of the following:

- A rectangle on the left represents a Java program
- A rectangle on the right represents an iSeries server
- Three vertically aligned small circles between the rectangles represent AS400 objects created by the Java program that connect to the iSeries server

The Java program (the left rectangle) contains several kinds of objects that use the AS400 objects (the three small circles). Lines from these program objects connect to the AS400 objects. Each program object requires only one AS400 object to use. However, several program objects of various types can use a single AS400 object to connect to the iSeries.

Lines from the AS400 objects connect to services contained in the iSeries server (the right rectangle). The services appropriately mirror the program objects. The following list describes how the multiple program objects, three AS400 objects, and iSeries server services connect to each other:

- An integrated file system program object connects to the first AS400 object, which connects to a file server job
- Another integrated file system program object connects to the second AS400 object, which connects to a second file server job
- The third AS400 objects has multiple program objects connected to it: a third integrated file system object, two data queue objects, and a command call object. The AS400 object then connects to the following services on the iSeries server: a third file server job (for the integrated file system object), a data queue server job (for the data queue objects) and a command call server job (for the command call object).

Long description of Figure 2: Java program creating a single AS400 object and socket connection for the same iSeries server (rzahh552.gif)

found in IBM Toolbox for Java: Managing connections

This figure illustrates that having multiple program objects share a single AS400 object reduces the number of connections and the amount of resource used on the system.

Description

The figure is composed of the following:

- A rectangle on the left represents a Java program
- A rectangle on the right represents an iSeries server
- A small circle between the rectangles represents a single AS400 object created by the Java program that connects to the iSeries server

The Java program (the left rectangle) contains several kinds of objects that use the AS400 object. Lines from these program objects connect to the AS400 object (the small circle).

Lines from the AS400 object connect to services contained in the iSeries server (the right rectangle). Each service represents a separate connection to the server. The services appropriately mirror the program objects:

- A data queue program object uses the AS400 object, which connects to a data queue server job
- Two integrated file system objects uses the AS400 object, which connects to a single file server job
- A command call object uses the AS400 object, which connects to a command call server job

All these program objects use the same AS400 object, which creates separate connections for each job that runs on the server. In this case, because the integrated file system objects use the same AS400 object, they share the same connection to and job on the server.

Long description of Figure 3: Java program getting a connection from an AS400ConnectionPool to an iSeries server (rzahh506.gif)

found in IBM Toolbox for Java: Managing connections

This figure illustrates how you can use a connection pool to manage your connections to an iSeries server.

Description

The figure is composed of the following:

- A rectangular image on the left represents a Java application
- A picture of an iSeries server on the right represents the server to which the Java application wants to connect
- An oval between the images on the left and right represents an AS400ConnectionPool object

Inside the AS400ConnectionPool (the oval) are multiple connections. Each connection is an AS400 object, which is pictured as a small image of an iSeries server with a lightning bolt on it:

- A circular arrow connects the Java program (the rectangle on the left) to the connection pool (the oval).
- A single line connects the iSeries server (the image on the right) to the connection pool (the oval).

The top half of the circular arrow points from the AS400ConnectionPool (the oval) to the Java application (the rectangular image on the left). On the arrow is one of the connections from the pool, and one of the connection images in the connection pool is faded. This represents a connection from the connection pool being requested and used by the Java application.

The bottom half of the circular arrow points from the Java application to the connection pool. This represents the Java application returning the connection to the connection pool.

The single line that connects the iSeries server (the image on the right) to the AS400ConnectionPool (the oval) represents socket connections to the server.

Long description of Figure 4: Single object using its own service for an instance of an AS400 object is disconnected (rzahh550.gif)

found in IBM Toolbox for Java: Managing connections

This figure illustrates how disconnecting the connection for a single AS400 object does not affect the connections of other AS400 objects.

Description

The figure is composed of the same elements as Figure 1:

- A rectangle on the left represents a Java program
- A rectangle on the right represents an iSeries server
- Three vertically aligned small circles between the rectangles represent AS400 objects created by the Java program that connect to the iSeries server

The Java program (the left rectangle) contains several kinds of objects that use the AS400 objects (the vertical line of three small circles). Lines from these program objects connect to the AS400 objects. Each program object requires only one AS400 object to use. However, several program objects of various types can use a single AS400 object to connect to the iSeries.

Lines from the AS400 objects connect to services contained in the iSeries server (the right rectangle). The services appropriately mirror the program objects. For a specific description of the different program objects, AS400 objects, and iSeries server services, see the description for Figure 1.

The connection for a single AS400 object is highlighted to show how disconnecting it does not affect the connections of the other AS400 objects. That connection consists of an integrated file system object, the AS400 object that it uses, the associated iSeries server service (a file server job), and the lines that connect them all. None of the other program objects, lines, AS400 objects, or services are highlighted.

Long description of Figure 5: All objects using the same service for an instance of an AS400 object are disconnected (rzahh551.gif)

found in IBM Toolbox for Java: Managing connections

This figure illustrates how disconnecting a service for a single AS400 object disconnects all objects that are sharing that service for that instance of the AS400 object.

Description

The figure is composed of the same elements as Figure 2:

- A rectangle on the left represents a Java program
- A rectangle on the right represents an iSeries server
- A small circle between the rectangles represents a single AS400 object created by the Java program that connects to the iSeries server

The Java program (the left rectangle) contains several kinds of objects that use the AS400 object (the small circle). Lines from these program objects connect to the AS400 object.

Lines from the AS400 object connect to services contained in the iSeries server (the right rectangle). The services appropriately mirror the program objects. For a specific description of the different program objects, AS400 object, and iSeries server services, see the description for Figure 2 .

Two integrated file system program objects use the AS400 objects, which connects to a file server job on the iSeries server. These two program objects, the associated service, and the lines that connect them are highlighted. Disconnecting the service affects only the highlighted elements, in effect disconnecting both program objects. None of the other program objects, services, connections, or the AS400 object itself are affected.

i5/OS Java virtual machine

The IBM Toolbox for Java classes run on the IBM Developer Kit for Java (i5/OS) Java virtual machine (JVM).

In fact, the classes run on any platform that supports the Java 2 Software Development Kit (J2SDK) specifications.

When you run IBM Toolbox for Java classes on the i5/OS JVM, do the following:

- Choose whether to use the i5/OS JVM or the IBM Toolbox for Java classes to access iSeries server resources when running in the i5/OS JVM.
- Check out Running IBM Toolbox for Java classes on the i5/OS JVM.
- Read about setting system name, user ID, and password in the i5/OS JVM.

For more information about iSeries server support for different Java platforms, see Support for multiple JDKs.

Comparing the i5/OS Java virtual machine and the IBM Toolbox for Java classes

You always have at least two ways to access an iSeries server resource when your Java program is running on the IBM Developer Kit for Java (i5/OS) Java virtual machine (JVM).

You can use either of the following interfaces:

- Facilities built into Java
- An IBM Toolbox for Java class

When deciding which interface to use, consider the following factors:

- **Location** - Where a program runs is the most important factor in deciding which interface set to use. Does the program do the following:
  - Run only on the client?
  - Run only on the server?
  - Run on both client and server, but in both cases the resource is an iSeries server resource?
  - Run on one i5/OS JVM and access resources on another iSeries server?
  - Run on different kinds of servers?

  If the program runs on both client and server (including an iSeries server as a client to a second iSeries server) and accesses only iSeries server resources, it may be best to use the IBM Toolbox for Java interfaces.

  If the program must access data on many types of servers, it may be best to use Java native interfaces.
- **Consistency / Portability** - The ability to run IBM Toolbox for Java classes on iSeries servers means that the same interfaces can be used for both client programs and server programs. When you have only one interface to learn for both client programs and server programs, you can be more productive.

  Writing to IBM Toolbox for Java interfaces makes your program less **server** portable, however.

  If your program must run to an iSeries server as well as other servers, you may find it better to use the facilities that are built into Java.
- **Complexity** - The IBM Toolbox for Java interface is built especially for easy access to an iSeries server resource. Often, the only alternative to using the IBM Toolbox for Java interface is to write a program that accesses the resource and communicates with that program through Java Native Interface (JNI).

You must decide whether it is more important to have better Java neutrality and write a program to access the resource, or to use the IBM Toolbox for Java interface, which is less portable.

- **Function** - The IBM Toolbox for Java interface often provides more function than the Java interface. For example, the IFSFileOutputStream class of the IBM Toolbox for Java licensed program has more function than the FileOutputStream class of java.io. Using IFSFileOutputStream makes your program specific to iSeries servers, however. You lose **server** portability by using the IBM Toolbox for Java class.

  You must decide whether portability is more important or whether you want to take advantage of the additional function.
- **Resource** - When running on the i5/OS JVM, many of the IBM Toolbox for Java classes still make requests through the host servers. Therefore, a second job (the server job) carries out the request to access a resource.

  This request may take more resource than a Java native interface that runs under the job of the Java program.
- **iSeries server as a client** - If your program runs on one iSeries server and accesses data on a second iSeries server, your best choice may be to use IBM Toolbox for Java classes. These classes provide easy access to the resource on the second iSeries server.

  An example of this is Data Queue access. The Data Queue interfaces of the IBM Toolbox for Java licensed program provide easy access to the data queue resource.

  Using the IBM Toolbox for Java also means your program works on both a client and server to access a data queue on an iSeries server. It also works when running on one iSeries server to access a data queue on another iSeries server.

  The alternative is to write a separate program (in C, for example) that accesses the data queue. The Java program calls this program when it needs to access the data queue.

  This method is more server-portable; you can have one Java program that handles data queue access and different versions of the program for each server you support.

Running IBM Toolbox for Java classes on the i5/OS Java virtual machine

There are some special considerations for running the IBM Toolbox for Java classes on the IBM Developer Kit for Java (i5/OS) Java virtual machine (JVM).

Command call

Two common ways to call a command are to use one of the following:

- The IBM Toolbox for Java CommandCall class
- The java.lang.Runtime.exec method

The CommandCall class generates a list of messages that are available to the Java program once the command completes. This list of messages is not available through java.lang.Runtime.exec().

The java.lang.Runtime.exec method is portable across many platforms, so if your program must access files on different types of servers, java.lang.Runtime.exec() is a better solution.

Integrated file system

Common ways to access a file in the integrated file system of the iSeries server:

- The IFSFile classes of the IBM Toolbox for Java licensed program
- The file classes that are a part of java.io

The IBM Toolbox for Java integrated file system classes have the advantage of providing more function than the java.io classes. The IBM Toolbox for Java classes also work in applets, and they do not need a method of redirection (such as iSeries Access for Windows) to get from a workstation to the server.

The java.io classes are portable across many platforms, which is an advantage. If your program must access files on different types of servers, java.io is a better solution.

If you use java.io classes on a client, you need a method of redirection (such as the iSeries Access for Windows) to get to the server file system.

JDBC

Two IBM-supplied JDBC drivers are available to programs running on the i5/OS JVM:

- The IBM Toolbox for Java JDBC driver
- The IBM Developer Kit for Java JDBC driver

The IBM Toolbox for Java JDBC driver is best to use when the program is running in a client/server environment.

The IBM Developer Kit for Java JDBC driver is best to use when the program is running on an iSeries server.

If the same program runs on both the workstation and the server, you should load the correct driver through a system property instead of coding the driver name into your program.

Program call

Two common ways to call a program are as follows:

- The ProgramCall class of the IBM Toolbox for Java
- Through a Java Native Interface (JNI) call

The ProgramCall class of the IBM Toolbox for Java licensed program has the advantage that it can call any iSeries server program.

You may not be able to call your iSeries server program through JNI. An advantage of JNI is that it is more portable across server platforms.

Setting system name, user ID, and password with an AS400 object in the i5/OS Java virtual machine

The AS400 object allows special values for system name, user ID, and password when the Java program is running on the IBM Developer Kit for Java (i5/OS) Java virtual machine (JVM).

When you run a program on the i5/OS JVM, be aware of some special values and other considerations:

- User ID and password prompting is disabled when the program runs on the server. For more information about user ID and password values in the server environment, see Summary of user ID and password values on an AS400 object.
- If system name, user ID, or password is not set on the AS400 object, the AS400 object connects to the current server by using the user ID and password of the job that started the Java program. **A password must be supplied when using record-level access while connecting to v4r3 and earlier machines. When connecting to a v4r4 or later machine, it can extend the signed-on user's password like the rest of the IBM Toolbox for Java components.**
- The special value, **localhost**, can be used as the system name. In this case, the AS400 object connects to the current server.
- The special value, ***current**, can be used as the user ID or password on the AS400 object. In this case, the user ID or password (or both) of the job that started the Java program is used. For more information about *current, see the following Notes.
- The special value, ***current**, can be used as the user ID or password on the AS400 object when the Java program is running on the i5/OS JVM of one iSeries server, and the program is accessing resources on

another iSeries server. In this case, the user ID and password of the job that started the Java program on the source system are used when connecting to the target system. For more information about *current, see the following Notes.

Notes:

- The Java program cannot set the password to "*current" if you are using record-level access and V4R3 or earlier. When you use record-level access, "localhost" is valid for system name and "*current" is valid for user ID; however, the Java program must supply the password.
- *current works only on systems running at Version 4 Release 3 (V4R3) and later. Password and user ID must be specified on system running on V4R2 systems.

Examples

The following examples show how to use the AS400 object with the i5/OS JVM.

Example: Creating an AS400 object when the i5/OS JVM is running a Java program

When a Java program is running in the i5/OS JVM, the program does not need to supply a system name, user ID, or password.

Note: You **must** supply a password when using record-level access.

If these values are not supplied, the AS400 object connects to the local system by using the user ID and password of the job that started the Java program.

When the program is running on the i5/OS JVM, setting the system name to **localhost** is the same as not setting the system name. The following example shows how to connect to the current server:

```
// Create two AS400 objects.  If the Java program is running in the
// i5/OS 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")
```

Example: Connecting to the current server with a different user ID and password from the program that started the job The Java program can set a user ID and password even when the program is running on the i5/OS JVM. These values override the user ID and password of the job that started the Java program.

In the following example, the Java program connects to the current server, but the program uses a user ID and password that differs from those of the job that started the Java program.

```
// 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")
```

Example: Connecting to a different server by using the user ID and password of the job that started the Java program

A Java program that is running on one server can connect to and use the resources of other iSeries servers.

If ***current** is used for user ID and password, the user ID and password of the job that started the Java program is used when the Java program connects to the target server.

In the following example, the Java program is running on one server, but uses resources from another server. The user ID and password of the job that started the Java program are used when the program connects to the second server.

```
// 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")
```

Summary of user ID and password values on an AS400 object:

The following table summarizes how the user ID and password values on an AS400 object are handled by a Java program running on a server compared to a Java program running on a client.

Values on AS400 object	Java program running on a server	Java program running on a client
System name, user ID, and password not set	Connect to the current server using the user ID and password of the job that started the program	Prompt for system, user ID, and password
System name = localhost	Connect to the current server using the user ID and password of the job that started the program	**Error:** localhost is not valid when the Java program is running on a client
System name = localhost User ID = *current		
System name = localhost User ID = *current Password ID = *current		
System name = ″sys″	Connect to server ″sys″ using the user ID and password of the job that started the program. ″sys″ can be the current server or another server	Prompt for user ID and password
System name = localhost User ID = ″UID″ Password ID = ″PWD″	Connect to the current server using the user ID and password specified by the Java program instead of the user ID and password of the job that started the program	**Error:** localhost is not valid when the Java program is not running on a client

Independent auxiliary storage pool (ASP)

An independent auxiliary storage pool (ASP) is a collection of disk units that you can bring online or take offline independent of the rest of the storage on a system.

Independent ASPs contain any of the following:

- one or more user-defined file systems
- one or more external libraries

Each Independent ASP contains all of the necessary system information associated with the data it contains. So, while the system is active, you can take the Independent ASP offline, bring it online, or switch between systems.

For more information, see Independent ASPs and User ASPs.

You can use the ″database name″ JDBC property or the setDatabaseName() method from the AS400JDBCDataSource class to specify the ASP to which you want to connect.

All other Toolbox for Java classes (IFSFile, Print, DataQueues, and so on) use the Independent ASP specified by the the job description of the user profile that connects to the server.

i5/OS optimization

The IBM Toolbox for Java licensed program is written in Java, so it runs on any platform with a certified Java virtual machine (JVM). The IBM Toolbox for Java classes function in the same way no matter where they run.

Additional classes come with i5/OS that enhance the behavior of the IBM Toolbox for Java when it is running on the iSeries JVM. Sign-on behavior and performance are improved when running on the iSeries JVM and connecting to the same iSeries server. i5/OS incorporated the additional classes starting at Version 4 Release 3.

Enabling the optimizations

IBM Toolbox for Java comes in two packages: as a separate licensed program and with i5/OS.

- Licensed Program 5722-JC1. The licensed program version of IBM Toolbox for Java ships files in the following directory:

  ```
  /QIBM/ProdData/http/public/jt400/lib
  ```

  These files do not contain i5/OS optimizations. Use these files if you want behavior consistent with running the IBM Toolbox for Java on a client.
- i5/OS. IBM Toolbox for Java is also shipped with i5/OS in directory

  ```
  /QIBM/ProdData/OS400/jt400/lib
  ```

  These files do contain the classes that optimize the IBM Toolbox for Java when running on the iSeries JVM.

  For more information see Note 1 in the information about Jar files.

Sign-on considerations

With the additional classes provided with i5/OS, Java programs have additional options for providing server (system) name, user ID and password information to the IBM Toolbox for Java.

When accessing a resource on an iSeries server, the IBM Toolbox for Java classes must have a system name, user ID and password.

- **When running on a client**, the system name, user ID and password are provided by the Java program, or the IBM Toolbox for Java retrieves these values from the user through a sign-on dialog.
- **When running on the iSeries Java virtual machine**, the IBM Toolbox for Java has one more option. It can send requests to the current (local) server using the user ID and password of the job that started the Java program.

With the additional classes, the user ID and password of the current job also can be used when a Java program that is running on one iSeries server accesses the resources on another iSeries server. In this case, the Java program sets the system name, then uses the special value ″*current″ for the user ID and password.

The Java program can only set the password to ″*current″ if you are using record-level access V4R4 or later. Otherwise, when you use record-level access, ″localhost″ is valid for system name and ″*current″ is valid for user ID; however, the Java program must supply the password.

A Java program sets system name, user ID, and password values in the AS400 object.

To use the user ID and password of the job, the Java program can use ″*current″ as user ID and password, or it can use the constructor that does not have user ID and password parameters.

To use the current server, the Java program can use ″localhost″ as the system name or use the default constructor. That is,

```
AS400 system = new AS400();
```

is the same as

```
AS400 system = new AS400("localhost", "*current", "*current");
```

Examples

The following examples show how to sign on to a server by using optimized classes.

Example: Signing on when using different AS400 constructors

Two AS400 objects are created in the following example. The two objects have the same behavior: they both run a command to the current server using the user ID and password of the job. One object uses the special value for the user ID and password, while the other uses the default constructor and does not set user ID or password.

```
                    // Create an AS400 object. Since the default
                    // constructor is used and system, user ID and
                    // password are never set, the AS400 object sends
                    // requests to the local iSeries using the job's
                    // user ID and password. If this program were run
                    // on a client, the user would be prompted for
                    // system, user ID and password.
AS400 sys1 = new AS400();

                    // Create an AS400 object. This object sends
                    // requests to the local iSeries using the job's
                    // user ID and password. This object will not work
                    // on a client.
AS400 sys2 = new AS400("localhost", "*current", "*current");

                    // Create two command call objects that use the
                    // AS400 objects.
CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

                    // Run the commands.
cmd1.run();
cmd2.run();
```

Example: Signing on by using the user ID and password of the current job

In the following example an AS400 object is created that represents a second iSeries server. Since ″*current″ is used, the job’s user ID and password from the iSeries server running the Java program are used on the second (target) server.

```
                    // Create an AS400 object. This object sends
                    // requests to a second iSeries 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();
```

Performance improvements

With the additional classes provided by i5/OS, Java programs running on the Java virtual machine for iSeries experience improved performance. Performance is improved in some cases because less communication function is used, and in other cases, an iSeries API is used instead of calling the server program.

Shorter download time

In order to download the minimum number of IBM Toolbox for Java class files, use the proxy server in combination with the AS400ToolboxJarMaker tool.

Faster communication

For all IBM Toolbox for Java functions except JDBC and integrated file system access, Java programs running on the Java virtual machine for iSeries will run faster. The programs run faster because less communication code is used when communicating between the Java program and the server program on the server that does the request.

JDBC and integrated file system access were not optimized because facilities already exist that make these functions run faster. When running on the iSeries, you can use the JDBC driver for iSeries instead of the JDBC driver that comes with the IBM Toolbox for Java. To access files on the server, you can use java.io instead of the integrated file system access classes that come with the IBM Toolbox for Java.

Directly calling i5/OS APIs

Performance of the following classes of the IBM Toolbox for Java is improved because these classes directly call i5/OS APIs instead of calling a server program to carry out the request:

- AS400Certificate classes
- CommandCall
- DataQueue
- ProgramCall
- Record-level database access classes
- ServiceProgramCall
- UserSpace

APIs are directly called only if the user ID and password match the user ID and password of the job running the Java program. To get the performance improvement, the user ID and password must match the user ID and password of the job that starts the Java program. For best results, use ″localhost″ for system name, ″*current″ for user ID, and ″*current″ for password.

Port mapping changes

The port mapping system has been changed, which makes accessing a port faster. Before this change, a request for a port would be sent to the port mapper. From there, the iSeries server would determine which port was available and return that port to the user to be accepted. Now, you can either tell the server which port to use or specify that the default ports be used. This option eliminates the wasted time of the server determining the port for you. Use the WRKSRVTBLE command to view or change the list of ports for the server.

For the port mapping improvement, a few methods have been added to AS400 class:

- getServicePort
- setServicePort
- setServicePortsToDefault

Language specific strings changes

Language-specific string files are now shipped within the IBM Toolbox for Java program as class files instead of property files. The iSeries server finds messages in class files faster than in property files. ResourceBundle.getString() now runs faster because the files are stored in the first place that the computer searches. Another advantage of changing to class files is that the server can find the translated version of a string faster.

Converters

Two classes allow faster, more efficient conversion between Java and the iSeries:

- Binary Converter: Converts between Java byte arrays and Java simple types.
- Character Converter: Converts between Java String objects and i5/OS code pages.

Also, the IBM Toolbox for Java now incorporates its own conversion tables for over 100 commonly used CCSIDs. Previously, the IBM Toolbox for Java either deferred to Java for nearly all text conversion. If Java did not possess the correct conversion table, IBM Toolbox for Java downloaded the conversion table from the server.

The IBM Toolbox for Java performs all text conversion for any CCSID of which it is aware. When it encounters an unknown CCSID, it attempts to let Java handle the conversion. At no point does the IBM Toolbox for Java attempt to download a conversion table from the server. This technique greatly reduces the amount of time it takes for an IBM Toolbox for Java application to perform text conversion. No action is required by the user to take advantage of this new text conversion; the performance gains all occur in the underlying converter tables.

Performance tip regarding the Create Java Program (CRTJVAPGM) command

If your Java application runs on the iSeries Java virtual machine (JVM), you can **significantly improve performance** if you create a Java program from an IBM Toolbox for Java .zip file or .jar file. Enter the **CRTJVAPGM** command on an iSeries command line to create the program. (See the online help information for the **CRTJVAPGM** command for more information.) By using the **CRTJVAPGM** command, you save the Java program that is created (and that contains the IBM Toolbox for Java classes) when your Java application starts. Saving the Java program that is created allows you to save startup processing time. You save startup processing time because the Java program on the server does not have to be re-created each time your Java application is started.

If you are using the V4R2 or V4R3 version of IBM Toolbox for Java, you cannot run the **CRTJVAPGM** command against the jt400.zip or jt400.jar file because it is too big; however, you may be able to run it against the jt400Access.zip file. At V4R3, IBM Toolbox for Java licensed program includes an additional file, jt400Access.zip. jt400Access.zip contains only the access classes, not the visual classes.

When you run Java applications on a V4R5 (or earlier) system, use jt400Access.zip. When you run Java applications on a V5R1 system, use jt400Native.jar. The **CRTJVAPGM** command has already been run against jt400Native.jar.

Java national language support

Java supports a set of national languages, but it is a subset of the languages that the server supports.

When a mismatch between languages occurs, for example, if you are running on a local workstation that is using a language that is not supported by Java, the IBM Toolbox for Java licensed program **may issue some error messages in English**.

Service and support for the IBM Toolbox for Java

Use the following resources for service and support.

IBM Toolbox for Java troubleshooting information Use this information to help you resolve problems when using IBM Toolbox for Java.

JTOpen/IBM Toolbox for Java forum Join the community of Java programmers who use IBM Toolbox for Java. This forum is an effective way to get assistance and advice from other Java programmers and sometimes from the IBM Toolbox for Java developers themselves

Server support Use the IBM Server support Web site to find out about the tools and resources that help you streamline the technical planning and support for your iSeries server.

Software support Use the IBM Software Support Services Web site to find out about the wide array of software support services offered by IBM.

Support services for the IBM Toolbox for Java, 5722-JC1, are provided under the usual terms and conditions for iSeries software products. Support services include program services, voice support, and consulting services. Contact your local IBM representative for more information.

Resolving IBM Toolbox for Java program defects is supported under program services and voice support, while resolving application programming and debugging issues is supported under consulting services.

IBM Toolbox for Java application program interface (API) calls are supported under consulting services unless any of the following are true:

- It is clearly a Java API defect, as demonstrated by re-creation in a relatively simple program.
- It is a question asking for documentation clarification.
- It is a question about the location of samples or documentation.

All programming assistance is supported under consulting services including those program samples provided in the IBM Toolbox for Java licensed program. Additional samples may be made available on the Internet at the iSeries home page on an unsupported basis.

Problem solving information is provided with the IBM Toolbox for Java Licensed Program Product. If you believe there is a potential defect in the IBM Toolbox for Java API, a simple program that demonstrates the error will be required.

Code examples

The following list provides links to entry points for many of the examples used throughout the IBM Toolbox for Java information.

Access classes
Beans
Commtrace classes
Graphical Toolbox
HTML classes
PCML
ReportWriter classes
Resource classes
RFML
Security classes
Servlet classes
Simple examples
Tips for programming
ToolboxMe for iSeries
Utility classes
Vaccess classes
XPCML

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Examples: Access classes

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java access classes.

AS400JPing

- Example: Using AS400JPing within a Java program

BidiTransform

- Example: Using the AS400BidiTransform class to transform bidirectional text

CommandCall

- Example: Using CommandCall to run a command on the server
- Example: Using CommandCall to prompt for the name of the server, command to run, and print the result

ConnectionPool

- Example: Using AS400ConnectionPool to create connections to the server

DataArea

- Example: Creating and writing to a decimal data area

Data conversion and description

- Examples: Using the FieldDescription, RecordFormat, and Record classes
- Example: Putting data on a queue
- Example: Reading data from a queue
- Example: Using AS400DataType classes with ProgramCall

DataQueue

- Example: How to create a DataQueue object, read data, and disconnect
- Example: Putting data on a queue
- Example: Reading data from a queue
- Example: Using KeyedDataQueue to put items on a queue
- Example: Using KeyedDataQueue to take items off a queue

Digital certificate
- Example: Listing digital certificates that belong to a user

EnvironmentVariable
- Example: Creating, setting, and getting environment variables

Exceptions
- Example: Catching a thrown exception, retrieving the return code, and displaying the exception text

FTP
- Example: Using the FTP class to Copy a set of files from a server directory
- Example: Using the AS400FTP class to copy a set of files from a directory

Integrated file system
- Examples: Using IFSFile
- Example: Using the IFSFile.listFiles() method to list the contents of a directory
- Example: Using IFSFile classes to copy files
- Example: Using IFSFile classes to list the contents of a directory
- Example: How to use IFSJavaFile instead of java.io.File
- Example: Using the IFSFile classes to list the contents of a directory on the server

JavaApplicationCall
- Example: Running a program on the server from the client that outputs ″Hello World!″

JDBC
- Example: Using the JDBC driver to create and populate a table
- Example: Using the JDBC driver to query a table and output its contents

Jobs
- Example: Retrieving and changing job information using the cache
- Example: Listing all active jobs
- Example: Printing all of the messages in the job log for a specific user
- Example: Listing the job identification information for a specific user
- Example: Getting a list of jobs on the server and listing the job status and the job identifier
- Example: Displaying messages in the job log for a job that belongs to the current user

Message queue
- Example: How to use the message queue object
- Example: Printing the contents of the message queue
- Example: Retrieving and printing a message
- Example: Listing the contents of the message queue
- Example: Using AS400Message with CommandCall
- Example: Using AS400Message with ProgramCall

NetServer
- Example: Using a NetServer object to change the name of the NetServer

Print
- Example: Asynchronously listing all spooled files using the PrintObjectListListener interface
- Example: Asynchronously listing all spooled files *without* using the PrintObjectListListener interface
- Example: Copying a spooled file using SpooledFile.copy()
- Example: Creating a spooled file from an input stream
- Example: Generating an SCS data stream using the SCS3812Writer class
- Example: Reading an existing spooled file
- Example: Reading and transforming spooled files
- Example: Synchronously listing all spooled files

Permission
- Example: Set the authority of an AS400 object

Program call
- Example: Using ProgramCall
- Example: Using ProgramCall to retrieve system status
- Example: Passing parameter data with a Program parameter object

QSYSObjectPathName
- Example: Building an integrated file system name
- Example: Using QSYSObjectPathName.toPath() to build an AS400 object name
- Example: Using QSYSObjectPathName to parse the integrated file system path name

Record-level access
- Example: Sequentially accessing a file
- Example: Using the record-level access classes to read a file
- Example: Using the record-level access classes to read records by key
- Example: Using the LineDataRecordWriter class

Service program call
- Example: Using ServiceProgramCall to call a procedure

SystemStatus
- Example: Use caching with the SystemStatus class

SystemPool
- Example: Setting the maximum faults size for SystemPool

SystemValue
- Example: Using SystemValue and SystemValueList

Trace
- Example: Using the Trace.setTraceOn() method
- Example: Preferred way to use Trace
- Example: Using component tracing

UserGroup
- Example: Retrieving a list of users
- Example: Listing all the users of a group

UserSpace

- Example: How to create a user space

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Using CommandCall

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// Command call example.  This program prompts the user
// for the name of the server and the command to run, then
// prints the result of the command.
//
// This source is an example of IBM Toolbox for Java "CommandCall"
//
////////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class CommandCallExample extends Object
{
   public static void main(String[] parmeters)
   {

      // Created a reader to get input from the user
      BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);



      // Declare variables to hold the system name and the command to run
      String systemString  = null;
      String commandString = null;

      System.out.println( " " );



      // Get the system name and the command to run from the user
      try
      {
         System.out.print("System name: ");
         systemString = inputStream.readLine();

         System.out.print("Command: ");
         commandString = inputStream.readLine();
```

```
      }
      catch (Exception e) {};

      System.out.println( " " );



      // Create an AS400 object.  This is the system we send the command to
      AS400 as400 = new AS400(systemString);



      // Create a command call object specifying the server that will
      // recieve the command.
      CommandCall command = new CommandCall( as400 );



      try
      {
         // Run the command.
         if (command.run(commandString))
             System.out.print( "Command successful" );
         else
             System.out.print( "Command failed" );



         // If messages were produced from the command, print them
         AS400Message[] messagelist = command.getMessageList();

         if (messagelist.length > 0)
         {
             System.out.println( ", messages from the command:" );
             System.out.println( " " );
         }

         for (int i=0; i < messagelist.length; i++)
         {
            System.out.print  ( messagelist[i].getID() );
            System.out.print  ( ": " );
            System.out.println( messagelist[i].getText() );
         }
      }
      catch (Exception e)
      {
         System.out.println( "Command " + command.getCommand() + " did not run" );
      }

      System.exit(0);
   }
}
```

Example: Using AS400ConnectionPool

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// AS400ConnectionPooling example.  This program uses an AS400ConnectionPool to
// create connections to an iSeries server.
// 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();
         }
      }
}
```

Examples: Using the FieldDescription, RecordFormat, and Record classes

The following examples show how you can use the FieldDescription, RecordFormat and Record classes with data queues.

Note: Read the Code example disclaimer for important legal information.

Example: Using the FieldDescription classes

You can use the FieldDescription classes can to describe the different types of data that make up an entry on a data queue. These examples assume the following format for entries on the data queue:

```
  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");
```

Using the RecordFormat class

You can use the RecordFormat class to describe the data that makes up the data queue entry.

Example: Defining RecordFormat and using it dynamically

The following example uses the RecordFormat class to describe the format of the data queue entry and then dynamically uses it to retrieve a record:

```
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();
```

Example: Defining RecordFormat statically

The following example defines the record format statically, which allows many programs to use the format without coding the record format multiple times.

```
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);
  }
}
```

Example: Using RecordFormat statically

The following example shows how a Java program can use a statically defined RecordFormat:

```
MessageEntryFormat entryFormat = new MessageEntryFormat();
// Get a record based on the format of the entries on the data queue
Record rec = entryFormat.getNewRecord();
```

Using the Record class

You can use the Record class to access individual fields of data queue entries.

Example: Using a generic Record object

```
// 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);
```

Example: Using a specific Record object

You can also statically define and use a Record specific to the format of this data queue, which allows you to provide get() and set() methods for the fields that are more meaningfully named than getField() and setField(). Also, by using the statically defined specific Record, you can return basic Java types instead of objects, and you can identify the return type for your user.

Note that this example must explicitly cast the correct Java object.

```
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
}
```

Example: Returning a new MessageEntryRecord

We need to override the getNewRecord() method in the MessageEntryFormat class (in the example above) in order to return a new MessageEntryRecord. To override the method, add the following to the MessageEntryFormat class:

```
public Record getNewRecord(byte[] data)
{
  Record r = new MessageEntryRecord();
  r.setContents(data);
  return r;
}
```

After adding the new getNewRecord() method, you can use the MessageEntryRecord to interpret the data queue entry:

```
// 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);
```

Example: Using DataQueue classes to put data on a queue

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// Data Queue example.  This program uses the DataQueue class to put
// records on a data queue.
//
// This example uses the Record and Record format classes to put data
// on the queue.  String data is converted from Unicode to ebcdic
```

```
// and numbers are converted from Java to the server format.  Since data
// is converted the data queue entries can be read by a server program
// or a iSeries Access for Windows program as well as another Java program.
//
// This is the producer side of the producer/consumer example.  It puts work
// items on the queue for the consumer to process.
//
// Command syntax:
//    DQProducerExample system
//
////////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQProducerExample extends Object
{
   // Create a reader to get input from the user.
   static BufferedReader inputStream =
      new BufferedReader(new InputStreamReader(System.in),1);

   public static void main(String[] parameters)
   {
      System.out.println( " " );

      // if the system name was not specified, display help text and exit.
      if (parameters.length >= 1)
      {
         try
         {
             // The first parameter is the system that contains the data queue.
             String system = parameters[0];

             // Create an AS400 object for the server that has the data queue.
             AS400 as400 = new AS400(system);

             // Build a record format for the format of the data queue entry.
             // This format matches the format in the DQConsumer class.  A
             // record consists of:
             //    - a four byte number    -- the customer number
             //    - a four byte number    -- the part number
             //    - a 20 character string -- the part description
             //    - a four byte number    -- the number of parts in this order
             // First create the base data types.
             BinaryFieldDescription customerNumber =
                new BinaryFieldDescription(new AS400Bin4(), "CUSTOMER_NUMBER");

             BinaryFieldDescription partNumber =
                new BinaryFieldDescription(new AS400Bin4(), "PART_NUMBER");

             CharacterFieldDescription partName =
                new CharacterFieldDescription(new AS400Text(20, as400), "PART_NAME");

             BinaryFieldDescription quantity =
                new BinaryFieldDescription(new AS400Bin4(), "QUANTITY");

             // Build a record format and fill it with the base data types.
             RecordFormat dataFormat = new RecordFormat();
             dataFormat.addFieldDescription(customerNumber);
             dataFormat.addFieldDescription(partNumber);
             dataFormat.addFieldDescription(partName);
             dataFormat.addFieldDescription(quantity);

             // Create the library that contains the data queue
```

```
        // using CommandCall.
        CommandCall crtlib = new CommandCall(as400);
        crtlib.run("CRTLIB JAVADEMO");

        // Create the data queue object.
        DataQueue dq = new DataQueue(as400, "/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

        // Create the data queue just in case this is the first time this
        // program has run.  The queue already exists exception is caught
        // and ignored.
        try
        {
           dq.create(96);
        }
        catch (Exception e) {};

        // Get the first field of data from the user.
        System.out.print("Enter customer number (or 0 to quit): ");
        int customer = getInt();

        // While there is data to put on the queue.
        while (customer > 0)
        {
           // Get the rest of the data for this order from the user.
           System.out.print("Enter part number: ");
           int part = getInt();

           System.out.print("Enter quantity: ");
           int quantityToOrder = getInt();

           String description = "part " + part;

           // Create a record based on the record format.  The record
           // is empty now but will eventually contain the data.
           Record data = new Record(dataFormat);

           // Set the values we received from the user into the record.
           data.setField("CUSTOMER_NUMBER", new Integer(customer));
           data.setField("PART_NUMBER",     new Integer(part));
           data.setField("QUANTITY",        new Integer(quantityToOrder));
           data.setField("PART_NAME", description);

           // Convert the record into a byte array.  The byte array is
           // what is actually put to the data queue.
           byte [] byteData = data.getContents();

           System.out.println("");
           System.out.println("Writing record to the server ...");
           System.out.println("");

           // Write the record to the data queue.
           dq.write(byteData);

           // Get the next value from the user.
           System.out.print("Enter customer number (or 0 to quit): ");
           customer = getInt();
        }
     }
     catch (Exception e)
     {
        // If any of the above operations failed say the data queue
        // operation failed and output the exception.

        System.out.println("Data Queue operation failed");
        System.out.println(e);
     }
  }
```

```
      // Display help text when parameters are incorrect.
      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Parameters are not correct. Command syntax is:");
         System.out.println("");
         System.out.println(" DQProducter system");
         System.out.println("");
         System.out.println("Where");
         System.out.println("");
         System.out.println(" system = Server that has the data queue");
         System.out.println("");
         System.out.println("For example:");
         System.out.println("");
         System.out.println(" DQProducerExample mySystem");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }

   // This is the subroutine that gets a character string from the user
   // and converts it into an int.
   static int getInt()
   {
      int i = 0;
      boolean Continue = true;

      while (Continue)
      {
         try
         {
            String s = inputStream.readLine();

            i = (new Integer(s)).intValue();
            Continue = false;
         }
         catch (Exception e)
         {
            System.out.println(e);
            System.out.print("Please enter a number ==>");
         }
      }

      return i;
   }
}
```

Example: Using DataQueue classes to read entries off a data queue

Note: Read the Code example disclaimer for important legal information.

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

Data types example usage

You can use the AS400DataType classes with ProgramCall to supply data for program parameters and to interpret the data returned in program parameters.

Note: Read the Code example disclaimer for important legal information.

Example: Using AS400DataType classes with ProgramCall

The following example shows how to use AS400DataType classes by using ProgramCall to call the system API, QUSRMBRD ″Retrieve Member Description″. The QUSRMBRD API retrieves the description of a specific member in a database file. The tables following the example list the required QUSRMBRD parameters and the types of information that the example retrieves.

```
// 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);
```

The following table lists the required parameters for the QUSRMBRD API as used in the preceding example.

QUSRMBRD parameter	Input or output	Type	Description
Receiver variable	Output	Char(*)	Character buffer which will contain the information retrieved.
Length of receiver variable	Input	Bin(4)	Length of the character buffer provided for the receiver variable.
Format name	Input	Char(8)	Format specifying the type of information to be retrieved. Must be one of: • MBRD0100 • MBRD0200 • MBRD0300 The following example specifies MBRD0100.

QUSRMBRD parameter	Input or output	Type	Description
Qualified database file name	Input	Char(20)	The qualified file name. This is the file name blank padded to 10 characters followed by the library name blank padded to 10 characters. Special values of *CURLIB and *LIBL are allowed for the library name.
Database member name	Input	Char(10)	The name of the member blank padded to 10 characters. Special values of *FIRST and *LAST are allowed.
Override processing	Input	Char(1)	Whether overrides are to be processed. 0 indicates that overrides are not to be processed. This is the value we will specify.

The following table lists the type of information that the example retrieves (based on format MBRD0100, as specified in the preceding example):

Information retrieved	Type
Bytes returned	Bin(4)
Bytes available	Bin(4)
Database file name	Char(10)
Database file library name	Char(10)
Member name	Char(10)
File attribute (type of file: PF, LF, DDMF)	Char(10)
Source type (type of the source source member if this is a source file)	Char(10)
Creation date and time	Char(13)
Last source change date and time	Char(13)
Member text description	Char(50)
Source file (whether the file is a source file: 0=data file, 1=source file)	Char(1)

Example: Using KeyedDataQueue

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// 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;
   }
}
```

Example: Using KeyedDataQueue classes to read entries off a data queue

Note: Read the Code example disclaimer for important legal information.

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

Examples: Using IFSFile

Note: Read the Code example disclaimer for important legal information.

The following examples show how to use the IFSFile class:

- Example: Creating a directory
- Example: Using IFSFile exceptions to track errors
- Example: Listing files with a .txt extension
- "Example: Using the IFSFile listFiles() method to list the contents of a directory" on page 469

Example: Creating a directory

```
                     // Create an AS400 object.  This new
                     // directory will be created on this
                     // iSeries.
    AS400 sys = new AS400("mySystem.myCompany.com");

                     // Create a file object that
                     // represents the directory.
    IFSFile aDirectory = new IFSFile(sys, "/mydir1/mydir2/newdir");

                     // Create the directory.
    if (aDirectory.mkdir())
       System.out.println("Create directory was successful");

                     // Else the create directory failed.
    else
    {
                     // If the object already exists,
                     // find out if it is a directory or
                     // file, then display a message.
       if (aDirectory.exists())
       {
          if (aDirectory.isDirectory())
             System.out.println("Directory already exists");
          else
             System.out.println("File with this name already exists");
       }
       else
          System.out.println("Create directory failed");
    }

                     // Disconnect since I am done
                     // accessing files.
    sys.disconnectService(AS400.FILE);
```

Example: Using IFSFile exceptions to track errors

When an error occurs, the IFSFile class throws the ExtendedIOException exception. This exception contains a return code that indicates the cause of the failure. The IFSFile class throws the exception even when the java.io class that IFSFile duplicates does not. For example, the delete method from java.io.File returns a boolean to indicate success or failure. The delete method in IFSFile returns a boolean, but if the delete fails, an ExtendedIOException is thrown. The ExtendedIOException provides the Java program with detailed information about why the delete failed.

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

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

                  // Delete the file.
try
{
   aFile.delete();

                  // The delete was successful.
   System.out.println("Delete successful ");
}

                  // The delete failed. Get the return
                  // code out of the exception and
                  // display why the delete failed.
catch (ExtendedIOException e)
{
   int rc = e.getReturnCode();

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

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

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

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

Example: Listing files with a .txt extension

The Java program can optionally specify match criteria when listing files in the directory. Match criteria reduce the number of files that are returned by the server to the IFSFile object, which improves performance. The following example shows how to list files with extension .txt:

```
                  // 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");
```

Example: Using the IFSFile listFiles() method to list the contents of a directory

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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 on list.
                // Print the name of the current file

                System.out.print(directoryFiles[i].getName());


                // Pad the output so the columns line up

                for (int j = directoryFiles[i].getName().length(); j <18; j++)
                    System.out.print(" ");


                // Print the date the file was last changed.

                long changeDate = directoryFiles[i].lastModified();
                Date d = new Date(changeDate);
                System.out.print(d);
                System.out.print("  ");


                // Print if the entry is a file or directory

                System.out.print("   ");

                if (directoryFiles[i].isDirectory())
                    System.out.println("");
                else
                    System.out.println(directoryFiles[i].length());

            }
    }

    catch (Exception e)
    {
        // If any of the above operations failed say the list failed
        // and output the exception.

        System.out.println("List failed");
        System.out.println(e);
    }
}

// Display help text when parameters are incorrect.

else
{
    System.out.println("");
    System.out.println("");
    System.out.println("");
```

```
            System.out.println("Parameters are not correct.  Command syntax is:");
            System.out.println("");
            System.out.println("   IFSListFiles as400 directory");
            System.out.println("");
            System.out.println("Where");
            System.out.println("");
            System.out.println("   as400 = system that contains the files");
            System.out.println("   directory = directory to be listed");
            System.out.println("");
            System.out.println("For example:");
            System.out.println("");
            System.out.println("   IFSListFiles mySystem /dir1/dir2");
            System.out.println("");
            System.out.println("");
        }

        System.exit(0);
    }
}


////////////////////////////////////////////////////////////////////////////
//
// The directory filter class prints information from the file object.
//
// Another way to use the filter is to simply return true or false
// based on information in the file object.  This lets the mainline
// function decide what to do with the list of files that meet the
// search criteria.
//
////////////////////////////////////////////////////////////////////////////



class MyDirectoryFilter implements IFSFileFilter
{
    public boolean accept(IFSFile file)
    {
        try
        {
            // Keep this entry.  Returning true tells the IFSList object
            // to return this file in the list of entries returned to the
            // .list() method.

            return true;
        }

        catch (Exception e)
        {
            return false;
        }
    }
}
```

Example: Using IFS classes to copy a file from one directory to another

Note: Read the Code example disclaimer for important legal information.

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

Example: Using the IFS classes to list the contents of a directory

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////////
//
// 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;
      }
   }
}
```

Example: Using JDBCPopulate to create and populate a table

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// JDBCPopulate example.  This program uses the IBM Toolbox for Java JDBC driver to
// create and populate a table.
//
// Command syntax:
//    JDBCPopulate system collectionName tableName
//
// For example,
//    JDBCPopulate MySystem MyLibrary MyTable
//
////////////////////////////////////////////////////////////////////////////
```

```
import java.sql.*;

public class JDBCPopulate
{



    // Strings to be added in the WORD column of the table.
    private static final String words[]
        = { "One",      "Two",      "Three",    "Four",     "Five",
            "Six",      "Seven",    "Eight",    "Nine",     "Ten",
            "Eleven",   "Twelve",   "Thirteen", "Fourteen", "Fifteen",
            "Sixteen",  "Seventeen","Eighteen", "Nineteen", "Twenty" };



    public static void main (String[] parameters)
    {
        // Check the input parameters.
        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Usage:");
            System.out.println("");
            System.out.println("   JDBCPopulate system collectionName tableName");
            System.out.println("");
            System.out.println("");
            System.out.println("For example:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCPopulate MySystem MyLibrary MyTable");
            System.out.println("");
            return;
        }

        String system           = parameters[0];
        String collectionName   = parameters[1];
        String tableName        = parameters[2];

        Connection connection   = null;

        try {

            // Load the IBM Toolbox for Java JDBC driver.
            DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

            // Get a connection to the database.  Since we do not
            // provide a user id or password, a prompt will appear.
            //
            // Note that we provide a default schema here so
            // that we do not need to qualify the table name in
            // SQL statements.
            //
            connection = DriverManager.getConnection ("jdbc:as400://" + system + "/" + collectionName);

            // Drop the table if it already exists.
            try {
                Statement dropTable = connection.createStatement ();
                dropTable.executeUpdate ("DROP TABLE " + tableName);
            }
            catch (SQLException e) {
                // Ignore.
            }

            // Create the table.
            Statement createTable = connection.createStatement ();
            createTable.executeUpdate ("CREATE TABLE " + tableName
```

```
                + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
                + " SQUAREROOT DOUBLE)");

            // Prepare a statement for inserting rows.  Since we
            // execute this multiple times, it is best to use a
            // PreparedStatement and parameter markers.
            PreparedStatement insert = connection.prepareStatement ("INSERT INTO "
                + tableName + " (I, WORD, SQUARE, SQUAREROOT) " + " VALUES (?, ?, ?, ?)");

            // Populate the table.
            for (int i = 1; i <= words.length; ++i) {
                insert.setInt (1, i);
                insert.setString (2, words[i-1]);
                insert.setInt (3, i*i);
                insert.setDouble (4, Math.sqrt(i));
                insert.executeUpdate ();
            }

            // Output a completion message.
            System.out.println ("Table " + collectionName + "." + tableName + " has been populated.");
        }

        catch (Exception e) {
            System.out.println ();
            System.out.println ("ERROR: " + e.getMessage());
        }

        finally {

            // Clean up.
            try {
                if (connection != null)
                    connection.close ();
            }
            catch (SQLException e) {
                // Ignore.
            }
        }

        System.exit (0);
    }


}
```

Example: Using JDBCQuery to query a table

Note: Read the Code example disclaimer for important legal information.

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


}
```

Example: Using JobList to list job identification information

Note: Read the Code example disclaimer for important legal information.

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

Example: Using JobList to get a list of jobs

Note: Read the Code example disclaimer for important legal information.

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

Example: Using JobLog to display messages in the job log

Note: Read the Code example disclaimer for important legal information.

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

}
```

Example: Creating spooled files

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////
//
// 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;
}

}
```

Example: Creating SCS spooled files

This example uses the SCS3812Writer class to generate an SCS data stream and write it to a spooled file on the server.

This application can take the following arguments, or it can use the defined defaults:

- Name of the server to receive the spooled file.
- Name of the outqueue on the server to receive the spooled file.

Note: Read the Code example disclaimer for important legal information.

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

Example: Reading spooled files

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////
//
// 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
    }
```

Example: Reading and transforming spooled files

The following examples demonstrate how to set up a PrintParameterList to obtain different transformations when reading spooled file data. In the code segments that follow, assume a spooled file already exists on a server, and the *createSpooledFile()* method creates an instance of the SpooledFile class representing the spooled file.

Example of PrintObjectPageInputStream

Note: Read the Code example disclaimer for important legal information.

The following example shows how to create a PrintObjectPageInputStream object for reading pages of data formatted as GIF images. In this case, each page from the spooled file will be transformed into a GIF image. A GIF workstation customization object is used to specify the data transform.

```
// 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);
```

Example of PrintObjectTransformedInputStream

Note: Read the Code example disclaimer for important legal information.

The following example shows how to create a PrintObjectTransformedInputStream object for reading data formatted as TIFF. A TIFF (G4 compression) workstation customization object is used to specify the data transform.

```
// 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);
```

Example of PrintObjectTransformedInputStream using manufacturer type and model

Note: Read the Code example disclaimer for important legal information.

The following example shows how to create a PrintObjectTransformedInputStream object for reading data formatted for output to an ASCII printer. A manufacturer type and model of *HP4 is used to specify the data transform.

```
// 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);
```

Example: Listing spooled files asynchronously (using listeners)

Note: Read the Code example disclaimer for important legal information.

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

}
```

Example: Listing spooled files asynchronously (without using listeners)

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////
//
// Example that shows listing all spooled files on a system Asynchronously without
// using the PrintObjectListListener interface. After opening the list the caller
// can do some additional work before waiting for the list to complete.
//
////////////////////////////////////////////////////////////////////////
//
// This source is an example of IBM Toolbox for Java "PrintObjectList".
//
////////////////////////////////////////////////////////////////////////

import java.util.Enumeration;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.SpooledFileList;
import com.ibm.as400.access.SpooledFile;

public class NPExampleListSplfAsynch2 extends Object
{
    private AS400 system_;

    public NPExampleListSplfAsynch2(AS400 system)
    {
        system_ = system;
    }

    // list all spooled files on the system asynchronously
    public void listSpooledFiles()
    {
```

```
    try
    {
        String strSpooledFileName;
        int listed, size;

        if( system_ == null )
        {
            system_ = new AS400();
        }

        System.out.println(
           "Now receiving all spooled files Asynchronously without using a listener");

        SpooledFileList splfList = new SpooledFileList(system_);

        // set filters, all users, on all queues
        splfList.setUserFilter("*ALL");
        splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

        // open list, openAsynchronously() returns immediately
        // we have not added any listeners...
        splfList.openAsynchronously();

        System.out.println(" Do some processing before waiting...");

        // ... do some processing here while the list is being built....

        System.out.println(" Now wait for list to complete.");

        // wait for the list to complete
        splfList.waitForListToComplete();

        Enumeration enum = splfList.getObjects();

        // output the name of all objects on the list
        while( enum.hasMoreElements() )
        {
            SpooledFile splf = (SpooledFile)enum.nextElement();
            if (splf != null)
            {
                // output this spooled file's name
                strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                System.out.println(" spooled file = " + strSpooledFileName);
            }
        }
        // clean up after we are done with the list
        splfList.close();
    }

    catch( Exception e )
    {
        // ...handle any exceptions...
        e.printStackTrace();
    }
}

public static void main( String args[] )
{
    NPExampleListSplfAsynch2 list = new NPExampleListSplfAsynch2(new AS400());
    try{
        list.listSpooledFiles();
    }
    catch( Exception e )
    {
        e.printStackTrace();
```

```
        }
        System.exit(0);
    }
}
```

Example: Listing spooled files synchronously

Note: Read the Code example disclaimer for important legal information.

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

Example: Using ProgramCall

Note: Read the Code example disclaimer for important legal information.

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

Example: Using record-level access classes

Note: Read the Code example disclaimer for important legal information.

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

Example: Using record-level access classes to read records from a file

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// 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);

   }
}
```

Example: Using record-level access classes to read records by key

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// Record-Level Access example.  This program uses the record-level
// access classes to read records by key from a file on the server.
// The user will be prompted for the server name to which to run and
// the library in which to create file QCUSTCDTKY.
//
// Command syntax:
//    java RLKeyedFileExample
//
// This program will copy the records from the iSeries Access for Windows sample
// database file (QCUSTCDT in library QIWS) to file QCUSTCDTKY which has
// the same format as QIWS/QCUSTCDT but has set the CUSNUM field as the key
// for the file.
//
// This source is an example of IBM Toolbox for Java "Record-level access".
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.math.*;
import com.ibm.as400.access.*;

public class RLKeyedFileExample
{
  public static void main(String[] parameters)
  {

    // Created a reader to get input from the user
    BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

    // Declare variables to hold the system name, library, file and member names
    String systemName = "";
    String library = "";

    // Get the system name from the user
    System.out.println();
    try
    {
```

```
  System.out.print("System name: ");
  systemName = inputStream.readLine();

  System.out.print("Library in which to create file QCUSTCDTKY: ");
  library = inputStream.readLine();
}
catch(Exception e)
{
  System.out.println("Error obtaining user input.");
  e.printStackTrace();
  System.exit(0);
}

// Create AS400 object and connect for the record level access service.
AS400 system = new AS400(systemName);
try
{
  system.connectService(AS400.RECORDACCESS);
}
catch(Exception e)
{
  System.out.println("Unable to connect for record level access.");
  System.out.println("Check the readme file for special instructions regarding record
                     level access");
  e.printStackTrace();
  System.exit(0);
}

RecordFormat qcustcdtFormat = null;
try
{
  // Create the RecordFormat object for creating the file.  The record format for the new
  // file will be the same as the record format for file QIWS/QCUSTCDT.  However we will
  // make the CUSNUM field a key field.
  AS400FileRecordDescription recordDescription =
    new AS400FileRecordDescription(system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

  // There is only one record format for the file, so take the first (and only) element
  // of the RecordFormat array returned as the RecordFormat for the file.
  System.out.println("Retrieving record format of QIWS/QCUSTCDT...");
  qcustcdtFormat = recordDescription.retrieveRecordFormat()[0];
  // Indicate that CUSNUM is a key field
  qcustcdtFormat.addKeyFieldDescription("CUSNUM");
}
catch(Exception e)
{
  System.out.println("Unable to retrieve record format from QIWS/QCUSTCDT");
  e.printStackTrace();
  System.exit(0);
}

// Create the keyed file object that represents the
// file we will create on the server.  We use a QSYSObectPathName object
// to get the name of the file into the correct format.
QSYSObjectPathName fileName = new QSYSObjectPathName(library,
                                                     "QCUSTCDTKY",
                                                     "*FILE",
                                                     "MBR");
KeyedFile file = new KeyedFile(system, fileName.getPath());

try
{
  System.out.println("Creating file " + library + "/QCUSTCDTKY...");
  // Create the file using the qcustcdtFormat object
  file.create(qcustcdtFormat, "Keyed QCUSTCDT file");

  // Populate the file with the records contained in QIWS/QCUSTCDT
```

```
  copyRecords(system, library);

  // Open the file for read-only access.  Because we will be randomly
  // accessing the file, specify a blocking factor of 1.  The
  // commit lock level parameter will be ignored since commitment
  // control has not been started.
  file.open(AS400File.READ_ONLY,
            1,
            AS400File.COMMIT_LOCK_LEVEL_NONE);

  // Assume that we want to display the information for customers
  // 192837, 392859 and 938472
  // The CUSNUM field is a zoned decimal field of length 6 with
  // no decimal positions.  Therefore, the key field value is
  // represented with a BigDecimal.
  BigDecimal[] keyValues = {new BigDecimal(192837),
                            new BigDecimal(392859),
                            new BigDecimal(938472)};

  // Create the key for reading the records.  The key for a KeyedFile
  // is specified with an Object[]
  Object[] key = new Object[1];

  Record data = null;
  for (int i = 0; i < keyValues.length; i++)
  {
    // Setup the key for reading
    key[0] = keyValues[i];

    // Read the record for customer number keyValues[i]
    data = file.read(key);
    if (data != null)
    {
      // Display the record only if balance due is greater than
      // zero.  In that case display the customer name and
      // the balance due.  The following code pulls fields out
      // of the record by field name.  As the field is retrieved
      // from the record it is converted from the server format to
      // Java format.
      if (((BigDecimal)data.getField("BALDUE")).floatValue() > 0.0)
      {
        System.out.print((String) data.getField("INIT") + "  ");
        System.out.print((String) data.getField("LSTNAM") + "  ");
        System.out.println((BigDecimal) data.getField("BALDUE"));
      }
    }
  }

  // All done with the file
  file.close();

  // Get rid of the file from the user's system
  file.delete();
}
catch(Exception e)
{
  System.out.println("Unable to create/read from QTEMP/QCUSTCDT");
  e.printStackTrace();
  try
  {
    file.close();
    // Get rid of the file from the user's system
    file.delete();
  }
  catch(Exception x)
  {
  }
```

```
    }

    // All done with record level access; disconnect from the
    // record-level access server.
    system.disconnectService(AS400.RECORDACCESS);
    System.exit(0);
  }

  public static void copyRecords(AS400 system, String library)
  {
    // Use the CommandCall class to run the CPYF command to copy the records
    // in QIWS/QCUSTCDT to QTEMP/QCUSTCDT
    CommandCall c = new CommandCall(system, "CPYF FROMFILE(QIWS/QCUSTCDT) TOFILE("
                                    + library + "/QCUSTCDTKY) MBROPT(*REPLACE)");
    try
    {
      System.out.println("Copying records from QIWS/QCUSTCDT to "
                         + library + "/QCUSTCDTKY...");
      c.run();
      AS400Message[] msgs = c.getMessageList();
      if (!msgs[0].getID().equals("CPC2955"))
      {
        System.out.println("Unable to populate " + library + "/QCUSTCDTKY");
        for (int i = 0; i < msgs.length; i++)
        {
          System.out.println(msgs[i]);
        }
        System.exit(0);
      }
    }
    catch(Exception e)
    {
      System.out.println("Unable to populate " + library + "/QCUSTCDTKY");
      System.exit(0);
    }
  }
}
```

Example: Using UserList to list all of the user in a given group

Note: Read the Code example disclaimer for important legal information.

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


}
```

Examples: JavaBeans

This section lists the code examples that are provided throughout the IBM Toolbox for Java bean information.

- Example: Using listeners to print a comment when you connect and disconnect to the system and run commands
- Example: Using applets and IBM VisualAge for Java to create buttons that run commands

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: IBM Toolbox for Java bean code

The following example creates an AS400 object and a CommandCall object, and then registers listeners on the objects. The listeners on the objects print a comment when the server connects or disconnects and when the CommandCall object completes the running of a command.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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()
```

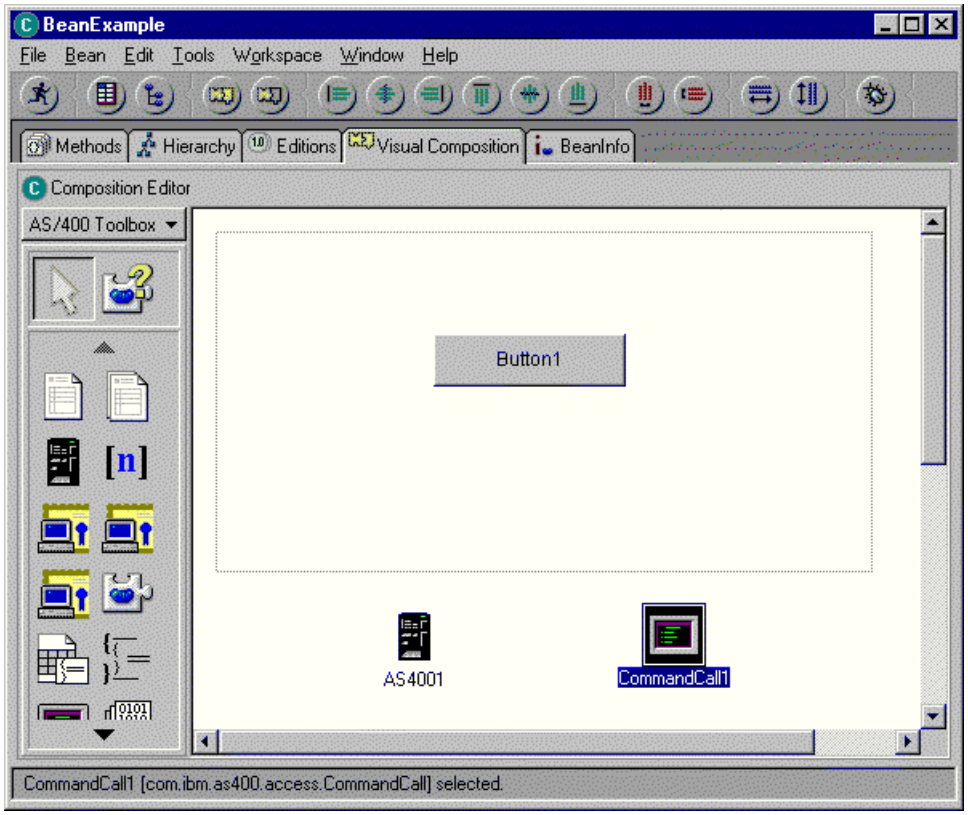
```
            {
                public void actionCompleted(ActionCompletedEvent event)
                {
                    System.out.println( "Command completed." );
                }
            }
        );
    }

    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);
    }
}
```

Example: Creating beans with a visual bean builder

This example uses the IBM VisualAge for Java Enterprise Edition V2.0 Composition Editor, but other visual bean builders are similar. This example creates an applet for a button that, when pressed, runs a command on the iSeries server.

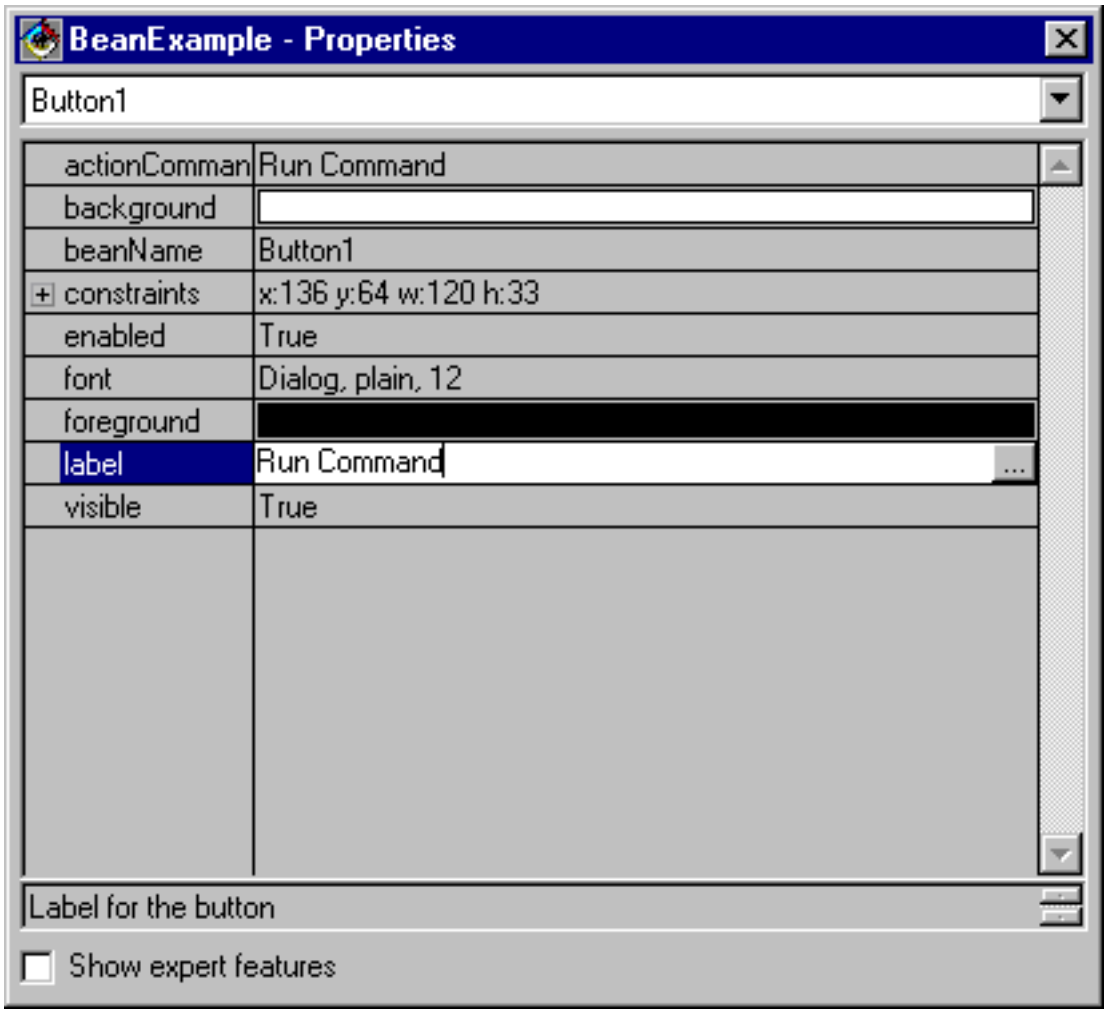
- Drag-and-drop a Button on the applet. (The Button can be found in the bean builder on the left side of the Visual Composition tab in Figure 1.)
- Drop a CommandCall bean and an AS400 bean outside the applet. (The beans can be found in the bean builder on the left side of the Visual Composition tab in Figure 1.)

Figure 1: VisualAge Visual Composition Editor window - gui.BeanExample



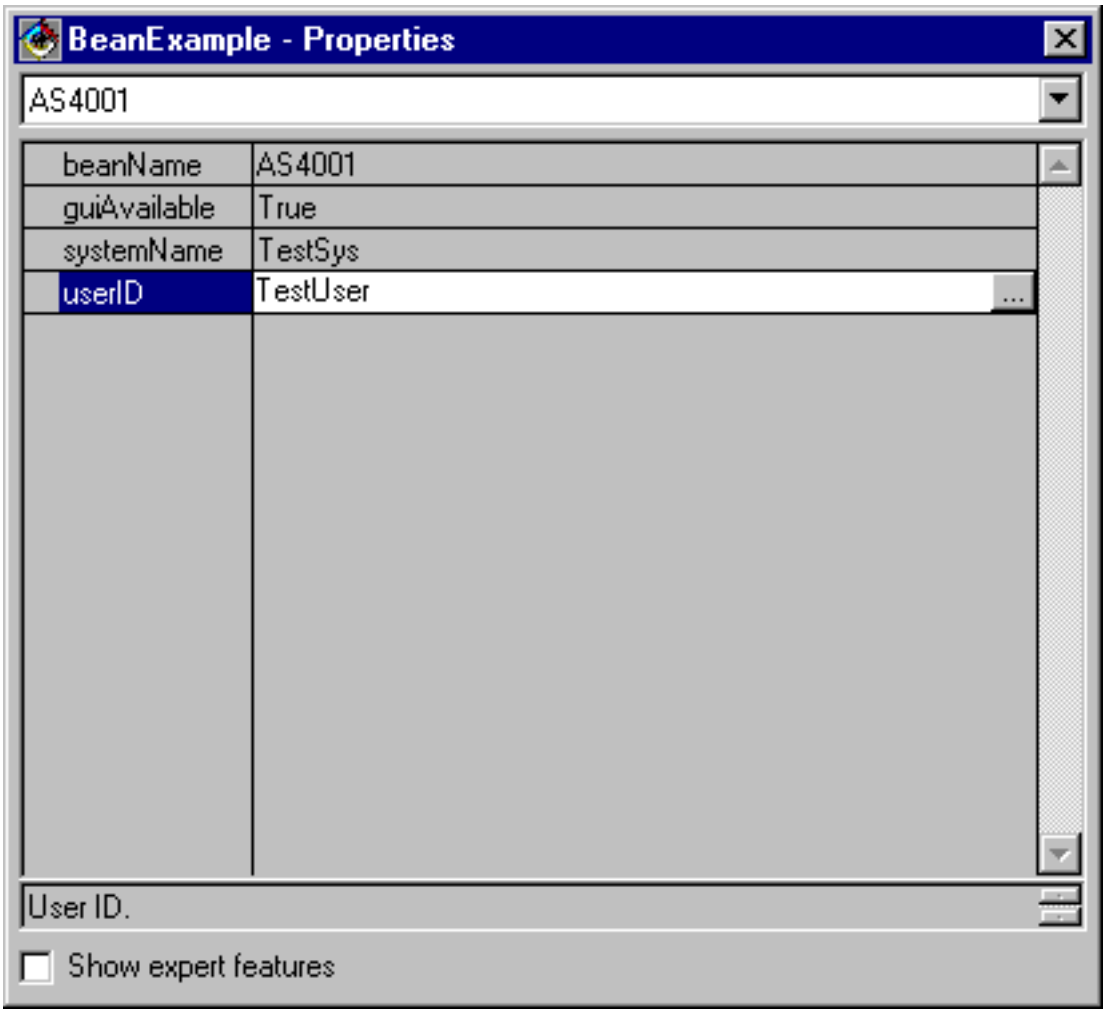
- Edit the bean properties. (To edit, select the bean and then right-click to display a window, which has Properties as an option.)
  - Change the label of the Button to **Run command**, as shown in Figure 2.

    Figure 2: Changing the label of the button to Run command



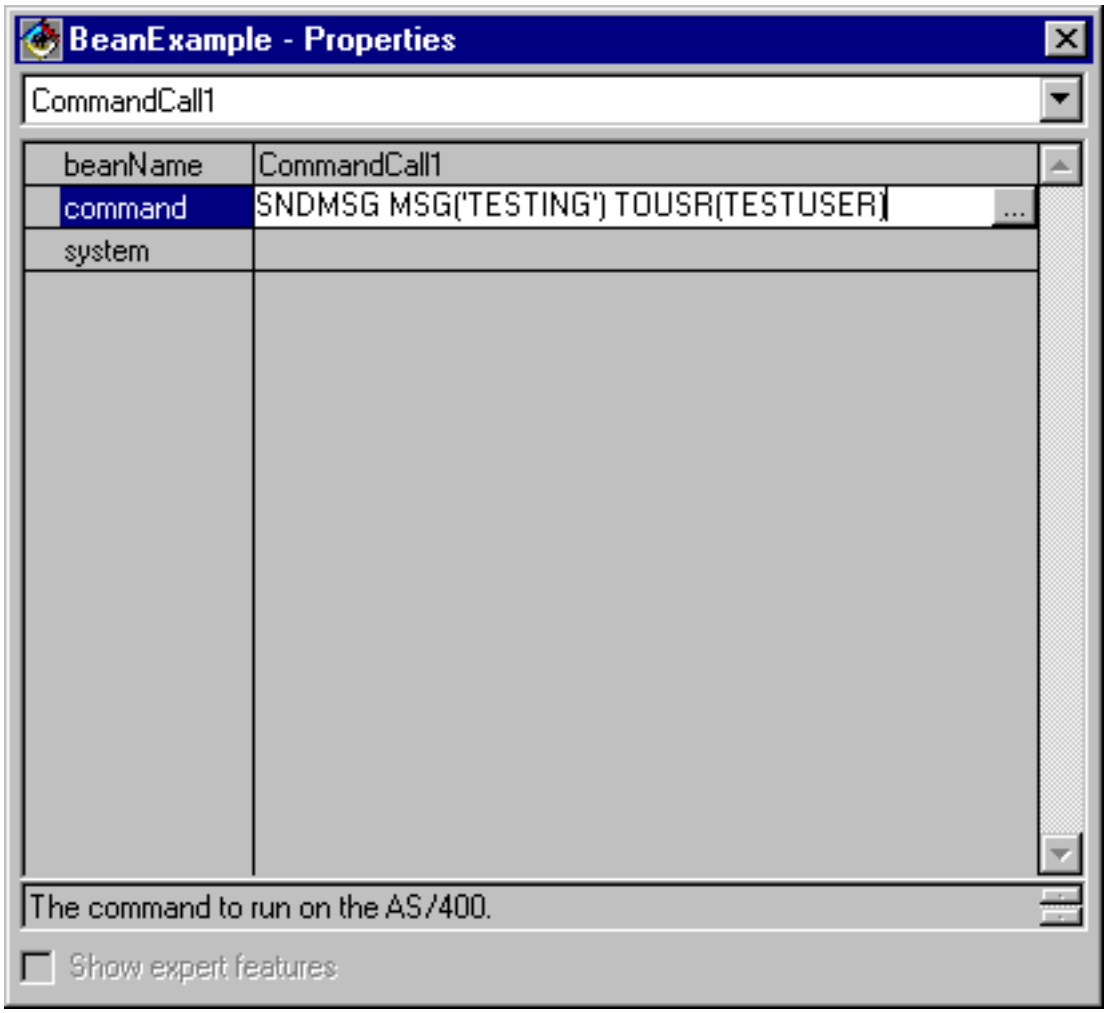
– Change the system name of the AS400 bean to **TestSys**
– Change the user ID of the AS400 bean to **TestUser**, as shown in Figure 3.

Figure 3: Changing the name of the user ID to TestUser



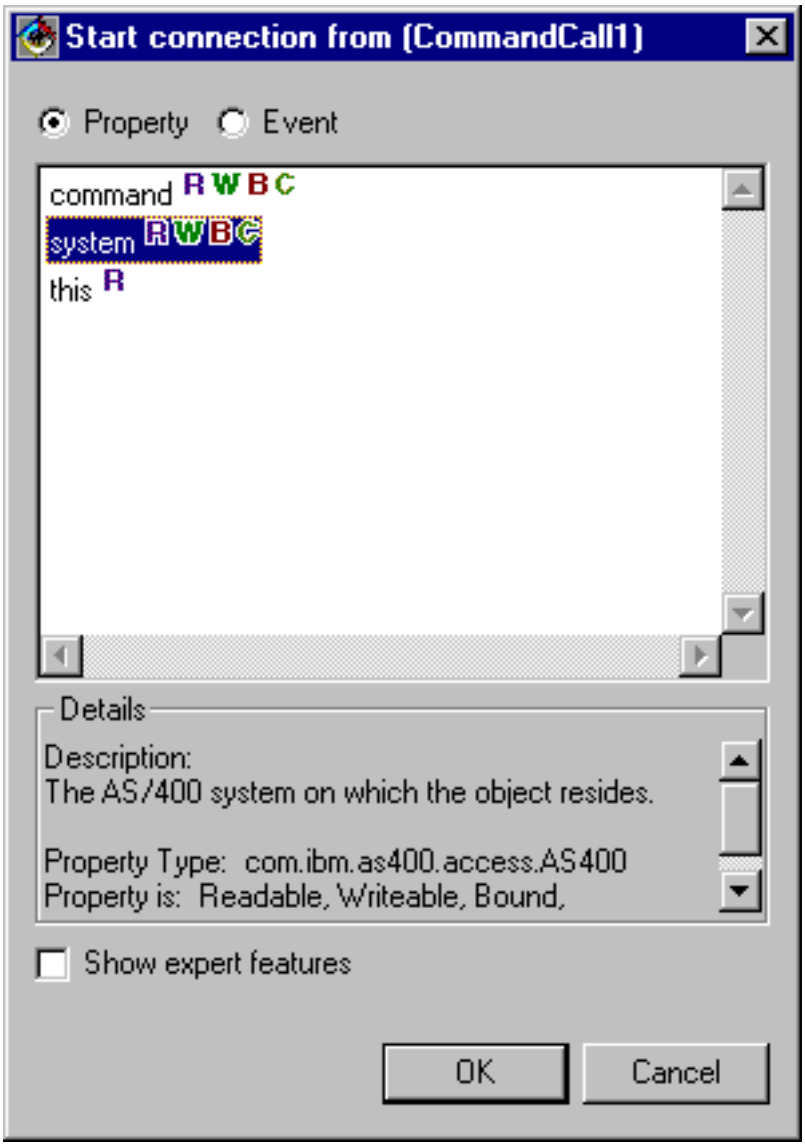
– Change the command of the CommandCall bean to **SNDMSG MSG('Testing') TOUSR('TESTUSER')**, as shown in Figure 4.

Figure 4: Changing the command of the CommandCall bean



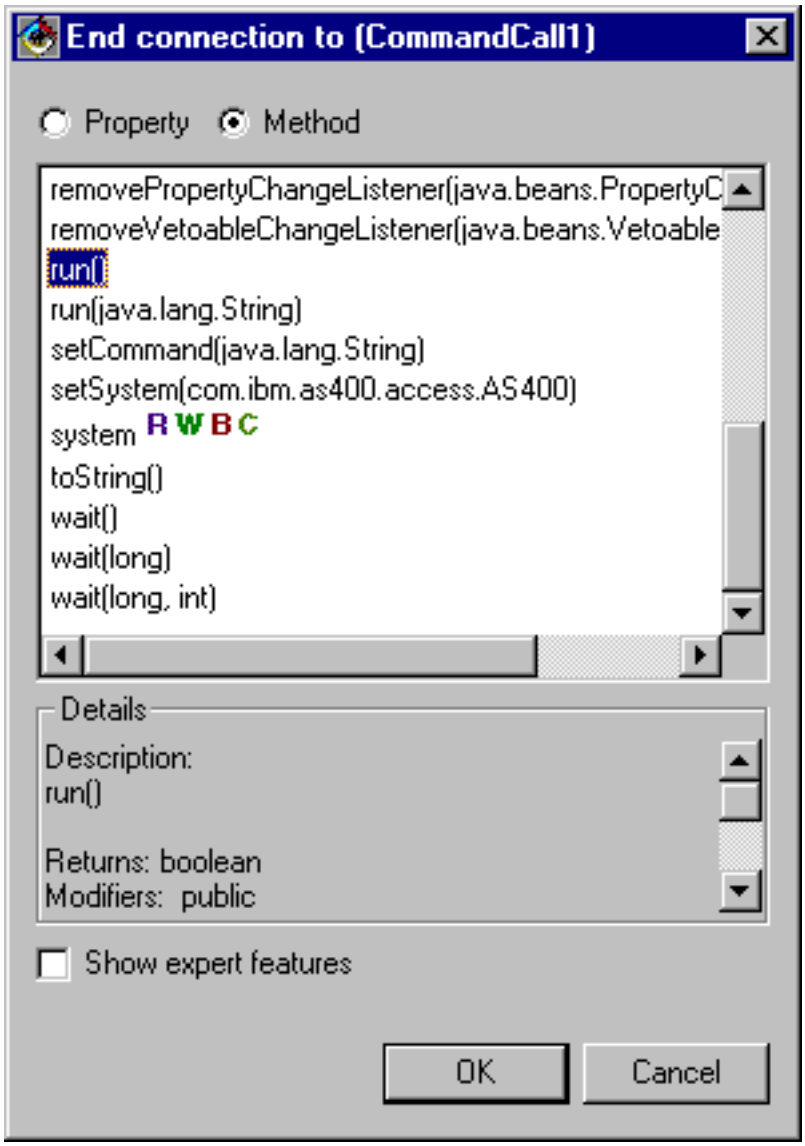
- Connect the AS400 bean to the CommandCall bean. The method you use to do this varies between bean builders. For this example, do the following:
  - Select the CommandCall bean and then click the right mouse button
  - Select **Connect**
  - Select **Connectable Features**
  - Select **system** from the list of features as shown in Figure 5.
  - Select the AS400 bean
  - Select **this** from the pop-up menu that appears over the AS400 bean

  Figure 5: Connecting AS400 bean to CommandCall bean



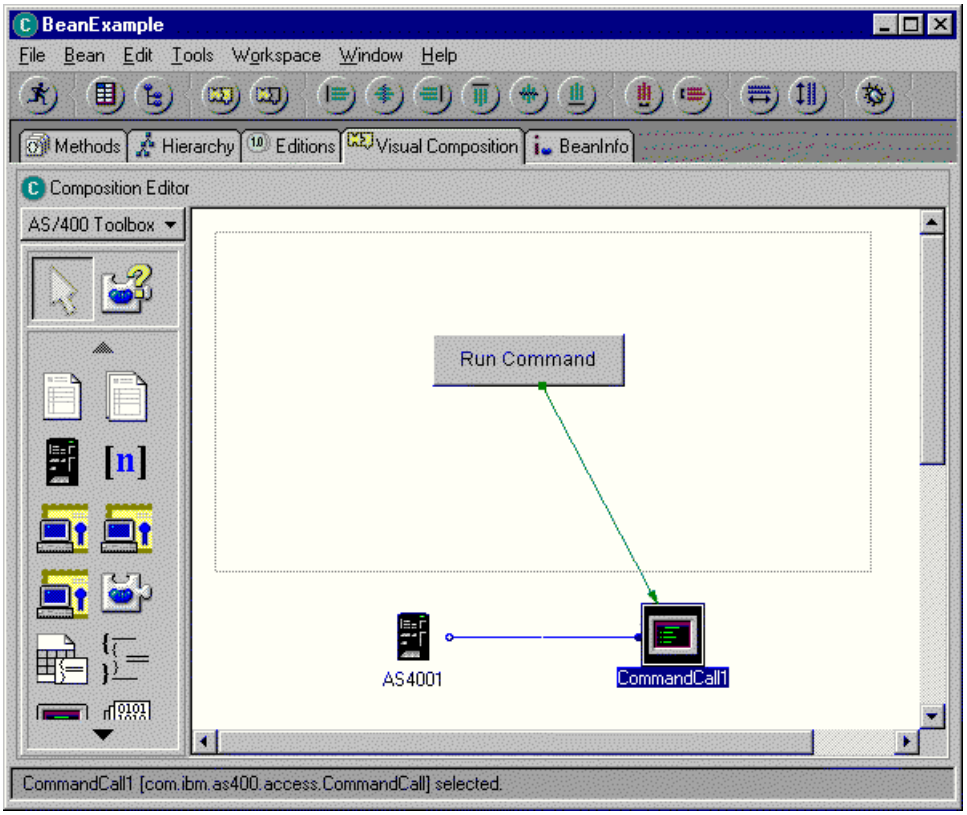
- Connect the button to the CommandCall bean.
  - Select the Button bean and then click the right mouse button
  - Select **Connect**
  - Select **actionPerformed**
  - Select the CommandCall bean
  - Select **Connectable Features** from the pop-up menu that appears
  - Select **run()** from the list of methods as shown in Figure 6.

    Figure 6: Connecting a method to a button



When you are finished, the VisualAge Visual Composition Editor window might look like Figure 7.

Figure 7: VisualAge Visual Composition Editor window - Finished bean example



Examples: Commtrace classes

This page links to the code example provided in the documentation of IBM Toolbox for Java commtrace classes.

- "Example: Using the commtrace classes" on page 182

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> **Code example disclaimer**
>
> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
>
> All programs contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Graphical Toolbox examples

Use these examples to show you how to implement the tools within Graphical Toolbox for your own UI programs.

- Construct and display a panel: Constructing a simple panel, which illustrates the basic features and operation of the Graphical Toolbox environment as a whole
- Create and display a panel: Creating and diplaying a panel when the panel and properties file are in the same directory

- Construct a fully-functional dialog: Constructing a fully-functioning dialog (after you have implemented the DataBeans that supply data to the panel and identified the attributes in the PDML)
- Size a panel using the dynamic panel manager: How the dynamic panel manager dynamically sizes the panel at runtime
- Editable combobox: Coding a data bean for an editable combobox

The following examples show you how the GUI Builder can help you to create a variety of GUI elements:

- Panels: Creating a sample panel and the data bean code that runs the panel
- Deckpanes: Creating a deckpane and what a final deckpane may look like
- Property sheets: Creating a property sheet and what a final property sheet may look like
- Split panes: Creating a split pane and what a final split pane may look like
- Tabbed panes: Creating a tabbed pane and what a final tabbed pane may look like
- Wizards: Creating a wizard and what the final product may look like
- Toolbars: Creating a tool bar and what a final tool bar may look like
- Menu bars: Creating a menu bar and what a final menu bar may look like
- Help: Generating a Help Document and split the Help Document into topic pages. Also, see Editing Help Documents generated by GUI builder
- Sample: Shows what a whole PDML program may look like, including panels, a property sheet, a wizard, select/deselect, and menu options.

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Example: Constructing a panel with the GUI Builder

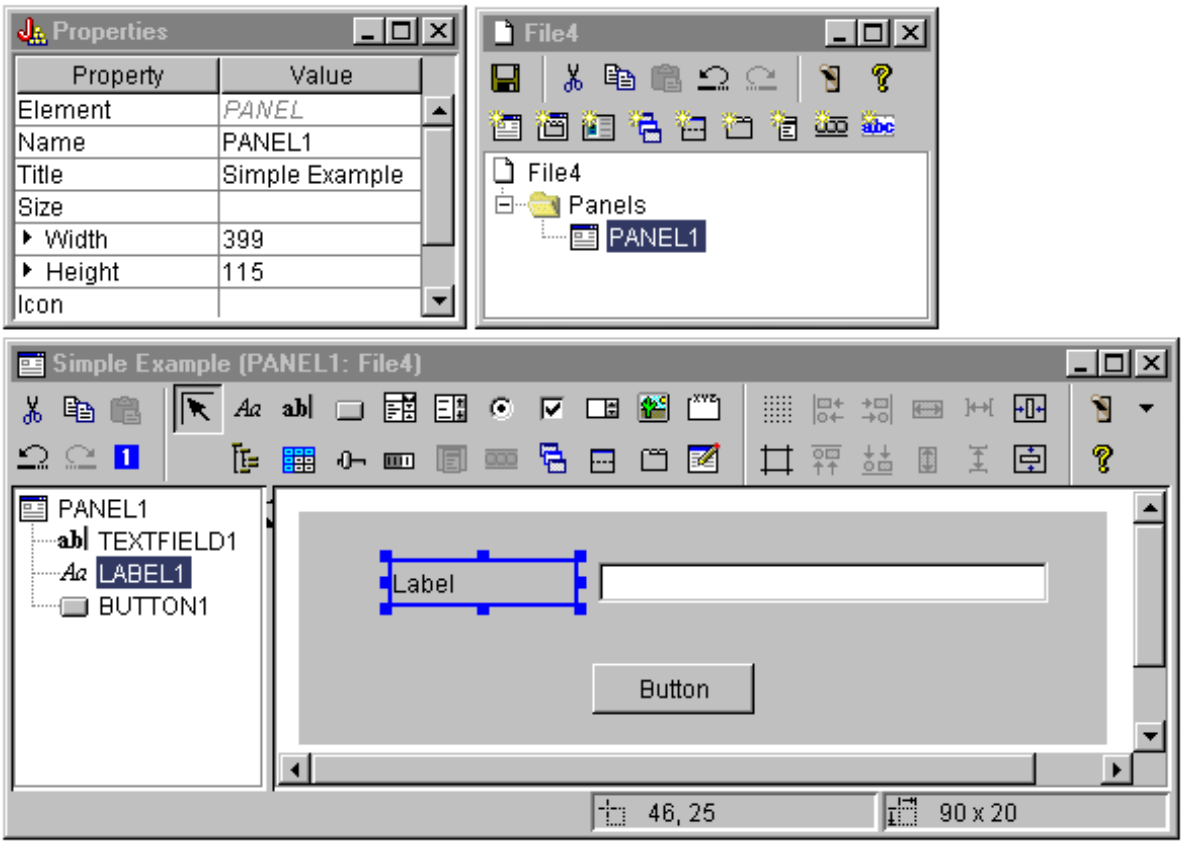
Note: Read the Code example disclaimer for important legal information.

This example demonstrates how to use the Graphical Toolbox by constructing a simple panel. It is an overview that illustrates the basic features and operation of the Graphical Toolbox environment. After showing you how to construct a panel, the example goes on to show you how to build a small Java application that displays the panel. In this example, the user enters data in a text field and clicks on the **Close** button. The application then echos the data to the Java console.

Constructing the panel

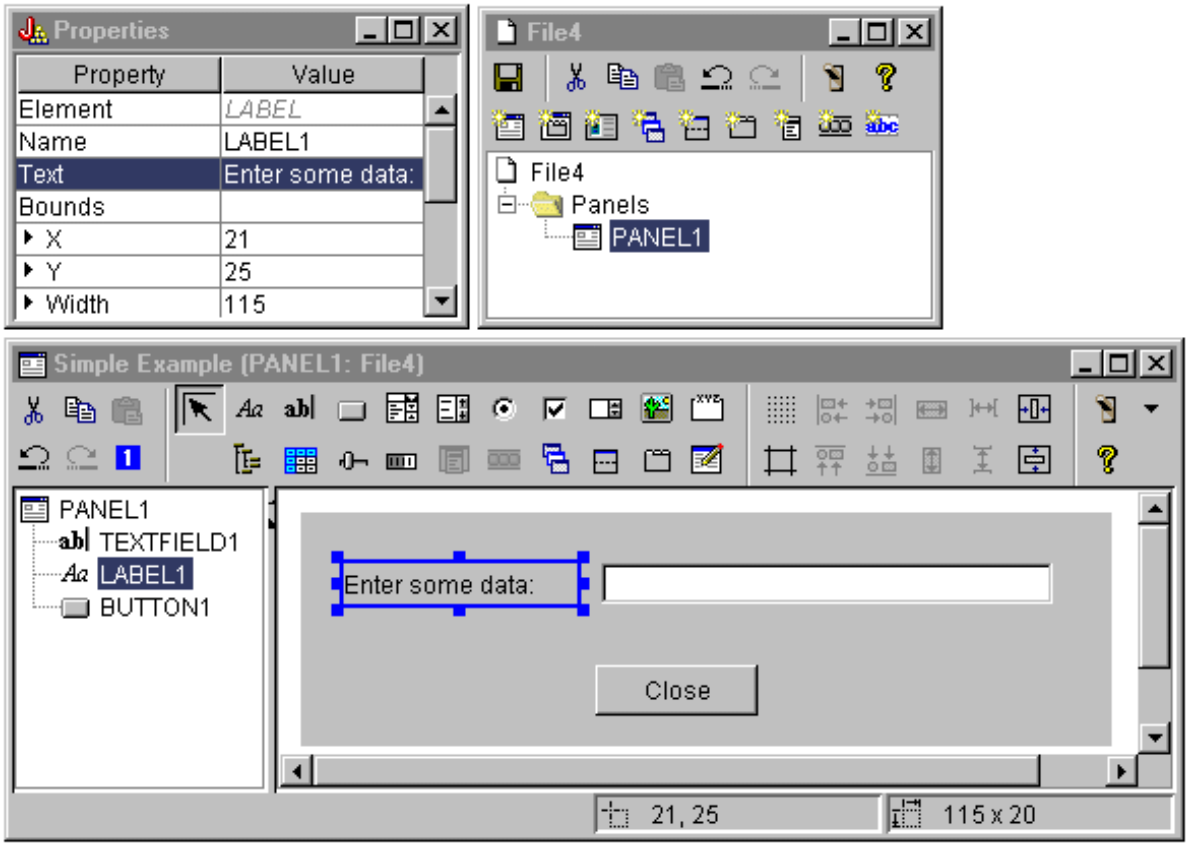
When you start the GUI Builder, the Properties and GUI Builder windows appear. Create a new file named ″MyGUI.pdml″. For this example, insert a new panel. Click the ″Insert Panel″ icon in File Builder window. Its name is ″PANEL1″. Change the title by modifying information in the Properties window; type ″Simple Example″ in the ″Title″ field. Remove the three default buttons by selecting them with your mouse and pressing ″Delete″. Using the buttons in the Panel Builder window, add the three elements shown in Figure 1: a label, a text field, and a pushbutton.

Figure 1: GUI Builder windows: Beginning to construct a panel



By selecting the label, you can change its text in the Properties window. In this example, the same has been done for the pushbutton, changing its text to "Close".

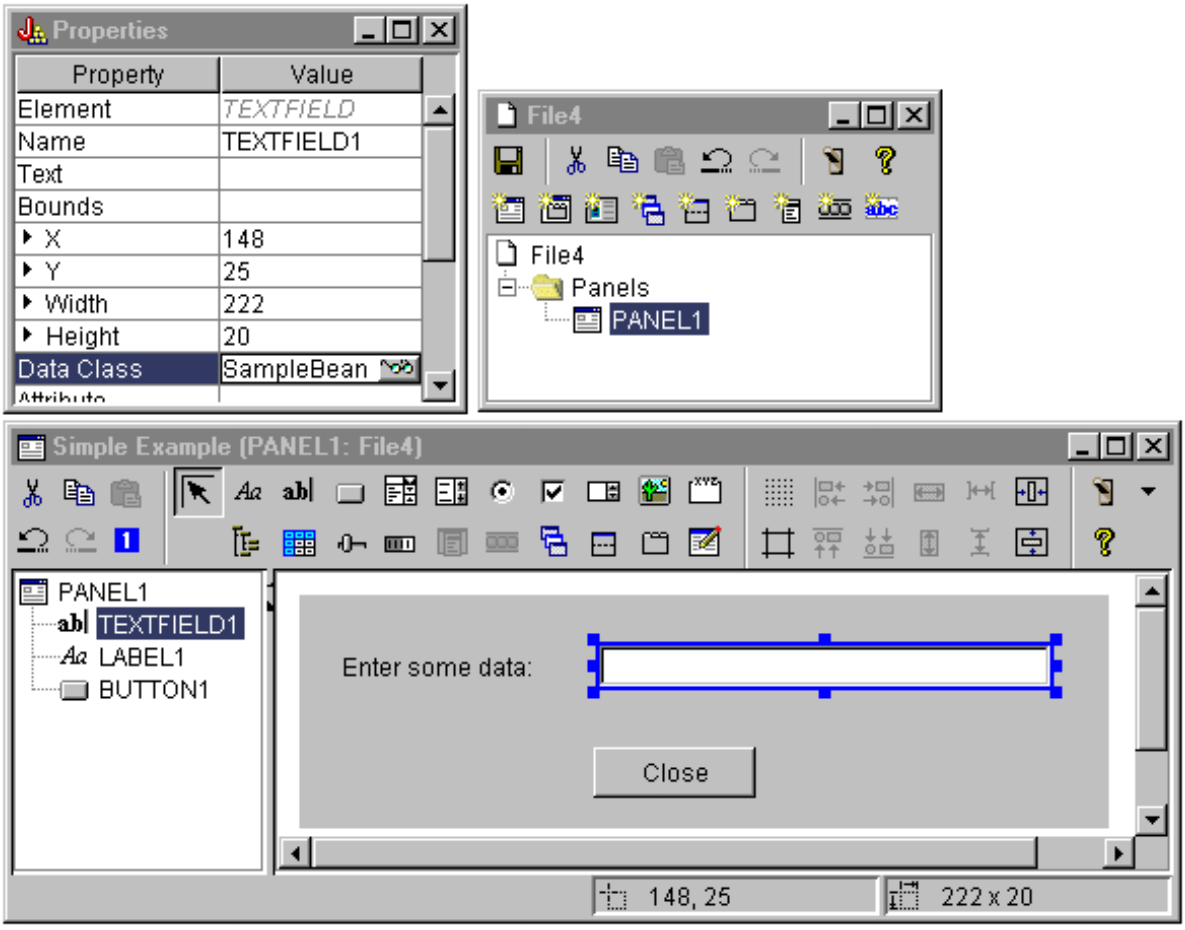
Figure 2: GUI Builder windows: Changing text in the Properties window



Text field

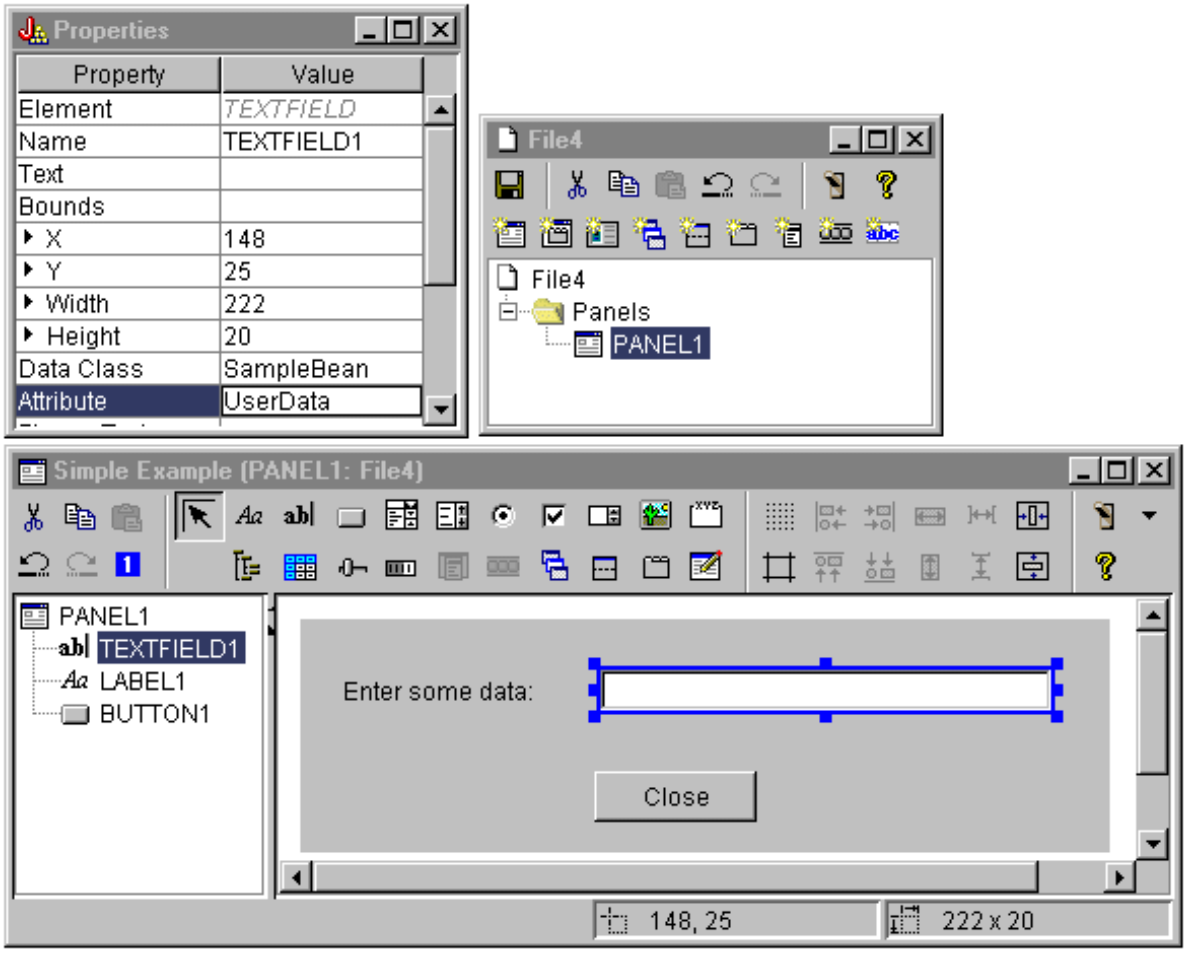
The text field will contain data and, therefore, you can set several properties that will allow the GUI Builder to perform some additional work. For this example, you set the Data Class property to the name of a bean class named **SampleBean**. This databean will supply the data for this text field.

Figure 3: GUI Builder windows: Setting the Data Class property



Set the Attribute property to the name of the bean property that will contain the data. In this case, the name is **UserData**.

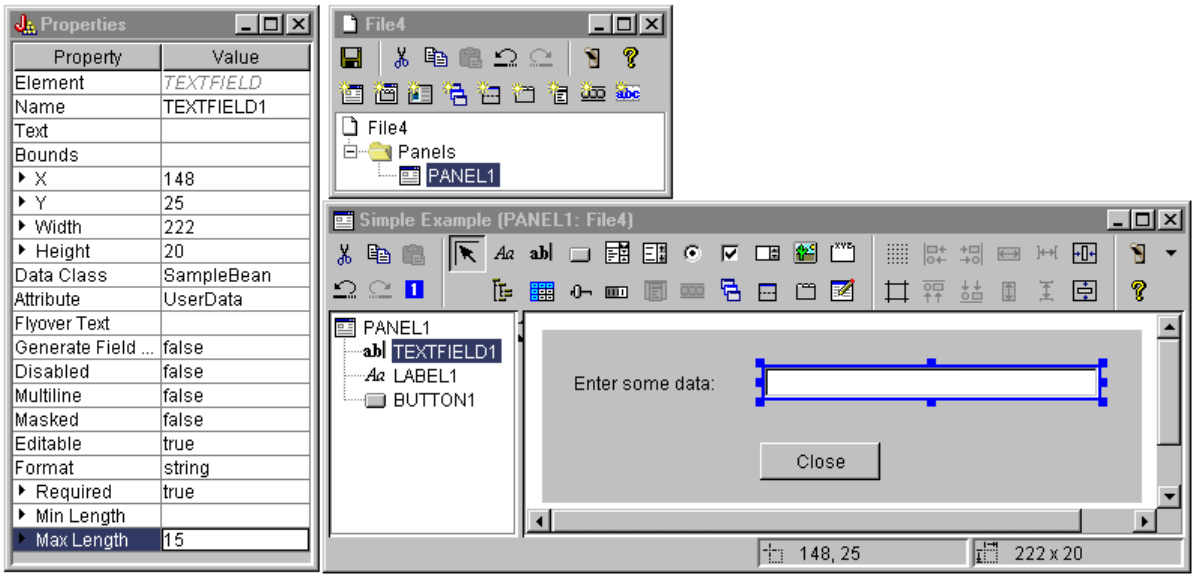
Figure 4: GUI Builder windows: Setting the Attribute property



Following the above steps binds the **UserData** property to this text field. At run-time, the Graphical Toolbox obtains the initial value for this field by calling **SampleBean.getUserData**. The modified value is then sent back to the application when the panel closes by calling **SampleBean.setUserData**.

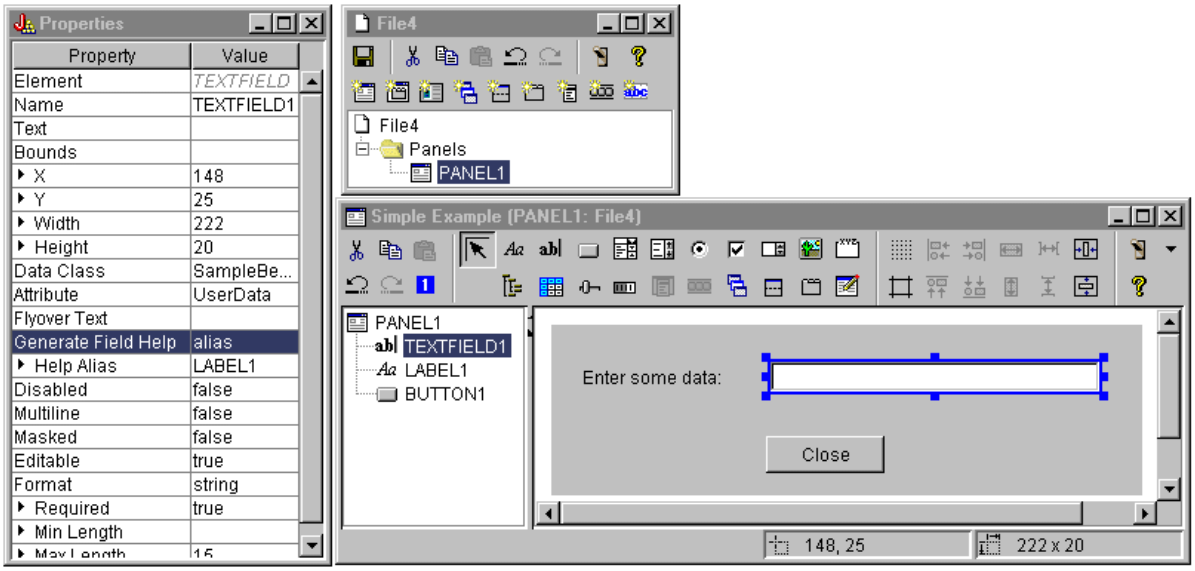
Specify that the user is required to supply some data, and that the data must be a string with a maximum length of 15 characters.

Figure 5: GUI Builder windows: Setting the maximum length of the text field



Indicate that the context-sensitive help for the text field will be the help topic associated with the label "Enter some data".

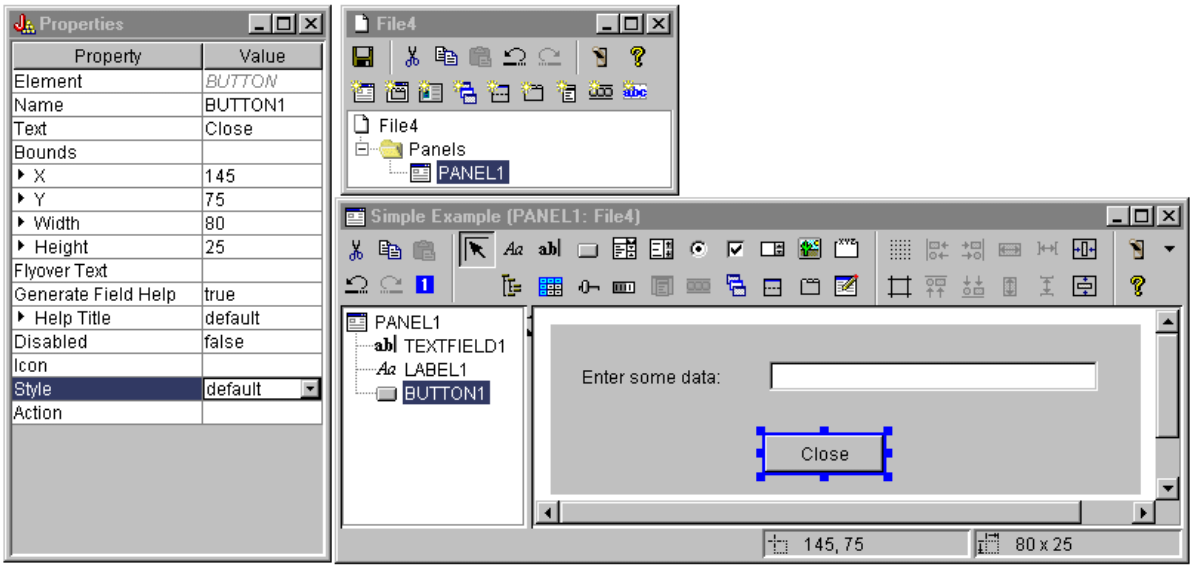
Figure 6: GUI Builder windows: Setting context-sensitive help for the text field



Button

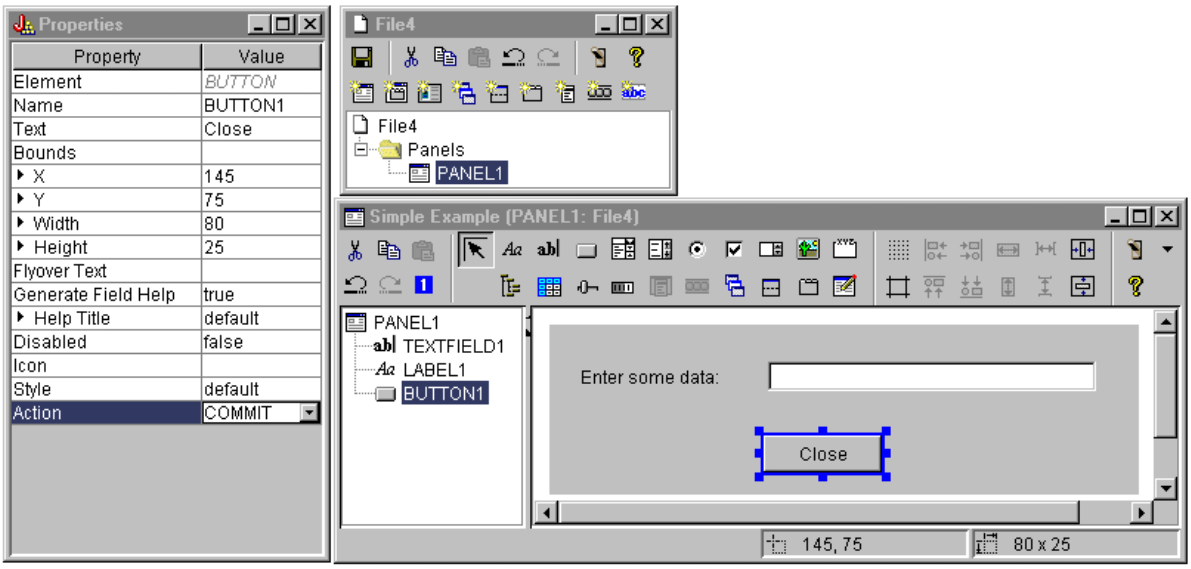
Modify the style property to give the button default emphasis.

Figure 7: GUI Builder windows: Setting the Style property to give the button default emphasis



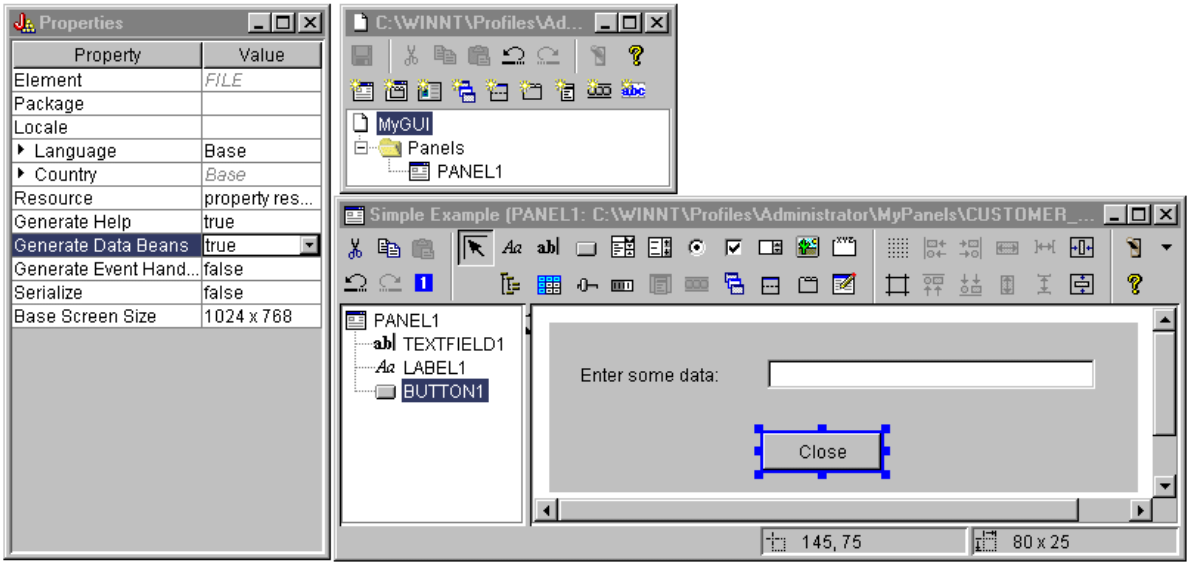
Set the ACTION property to COMMIT, which causes the `setUserData` method on the bean to be called when the button is selected.

Figure 8: GUI Builder windows: Setting the Action property to COMMIT



Before you save the panel, set properties at the level of the PDML file to generate both the online help skeleton and the Java bean. Then you save the file by clicking on the  icon in the GUI Builder window. When prompted, specify a file name of **MyGUI.pdml**.

Figure 9: GUI Builder windows: Setting properties to generate the online help skeleton and the Java bean



Generated files

After you save the panel definition, you can look at the files produced by the GUI Builder. **PDML file** Here is the content of **MyGUI.pdml** to give you an idea of how the Panel Definition Markup Language works. Because you use PDML only through the tools provided by the Graphical Toolbox, it is not necessary to understand the format of this file in detail:

```
<!-- 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>
```

Resource bundle

Associated with every PDML file is a resource bundle. In this example, the translatable resources were saved in a PROPERTIES file, which is called **MyGUI.properties**. Notice that the PROPERTIES file also contains customization data for the GUI Builder.

```
##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

The example also generated a Java source code skeleton for the JavaBean object. Here is the content of **SampleBean.java**:

```
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 ="";
    }
}
```

Note that the skeleton already contains an implementation of the gettor and settor methods for the `UserData` property. The other methods are defined by the `DataBean` interface and, therefore, are required.

The GUI Builder has already invoked the Java compiler for the skeleton and produced the corresponding class file. For the purposes of this simple example, you do not need to modify the bean implementation. In a real Java application you would typically modify the `load` and `save` methods to transfer data from an external data source. The default implementation of the other two methods is often sufficient. For more information, see the documentation on the **DataBean** interface in the javadocs for the PDML runtime framework.

Help file

The GUI Builder also creates an HTML framework called a Help Document. Help writers can easily manage help information by editing this file. For more information, see the following topics:

- Creating the Help Document
- Editing Help Documents generated by GUI builder

Constructing the application

Once the panel definition and the generated files have been saved, you are ready to construct the application. All you need is a new Java source file that will contain the main entry point for the application. For this example, the file is called **SampleApplication.java**. It contains the following code:

```
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);
    }
}
```

It is the responsibility of the calling program to initialize the bean object or objects by calling **load**. If the data for a panel is supplied by multiple bean objects, then each of the objects must be initialized before passing them to the Graphical Toolbox environment.

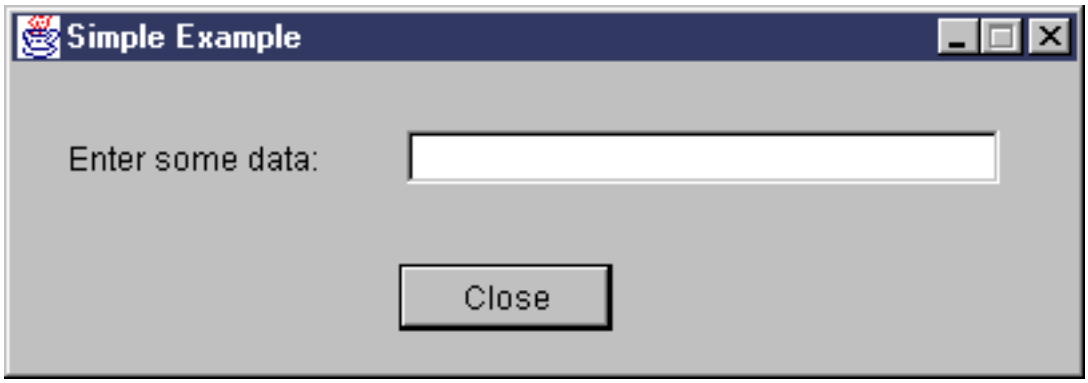
The class **com.ibm.as400.ui.framework.java.PanelManager** supplies the API for displaying standalone windows and dialogs. The name of the PDML file as supplied on the constructor is treated as a resource name by the Graphical Toolbox - the directory, ZIP file, or JAR file containing the PDML must be identified in the classpath.

Because a **Frame** object is supplied on the constructor, the window will behave as a modal dialog. In a real Java application, this object might be obtained from a suitable parent window for the dialog. Because the window is modal, control does not return to the application until the user closes the window. At that point, the application simply echoes the modified user data and exits.

Running the application

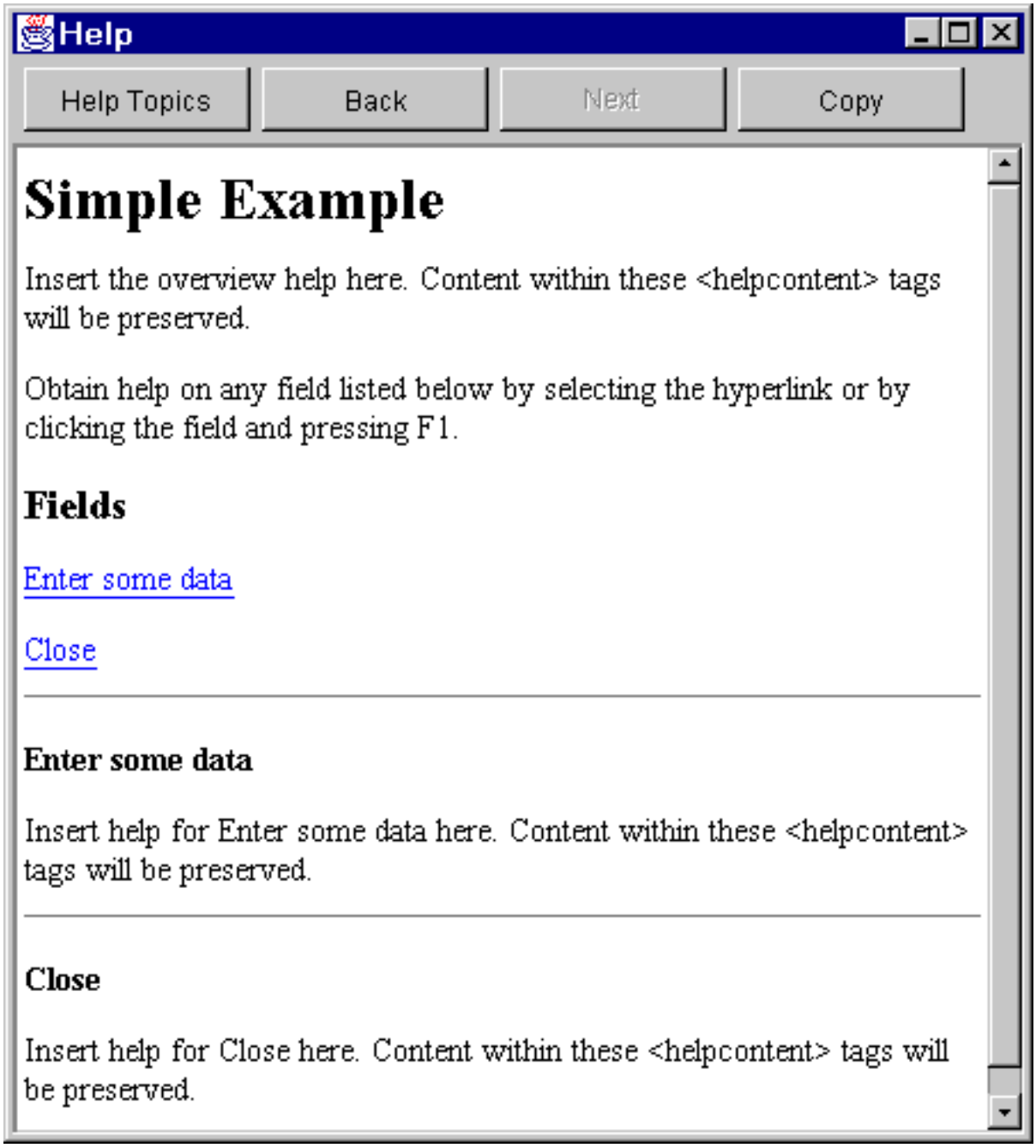
Here is what the window looks like when the application is compiled and run:

Figure 10: The Simple Example application window



If the user presses F1 while focus is on the text field, the Graphical Toolbox will display a help browser containing the online help skeleton that the GUI Builder generated.

Figure 11: The Simple Example online help skeleton



You can edit the HTML and add actual help content for the help topics shown.

If the data in the text field is not valid (for example, if the user clicked on the **Close** button without supplying a value), the Graphical Toolbox will display an error message and return focus to the field so that data can be entered.

Figure 12: Data Error message

For information about how to run this sample as an applet, see Using the Graphical Toolbox in a Browser.

Editable Comboboxes

When the bean generator creates a gettor and settor for an Editable ComboBox, by default it returns a String on the settor and takes a string parameter on the gettor. It can be useful to change the settor to take an Object class and the gettor to return an Object type. This allows you to determine the user selection using ChoiceDescriptors.

If a type of Object is detected for the gettor and settor, the system will expect either a ChoiceDescriptor or a type Object instead of a formatted string.

The following example assumes that Editable is an editable ComboBox that has either a Double value, uses a system value, or is not set.

```
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;
    }
}
```

Similarly, when a type of Object is detected for the gettor and settor, the system will return an Object which is either a ChoiceDescriptor containing the selected choice or a type Object.

```
     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 */ }
     }
```

Example: Using RecordListFormPane

Note: Read the Code example disclaimer for important legal information.

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


}
```

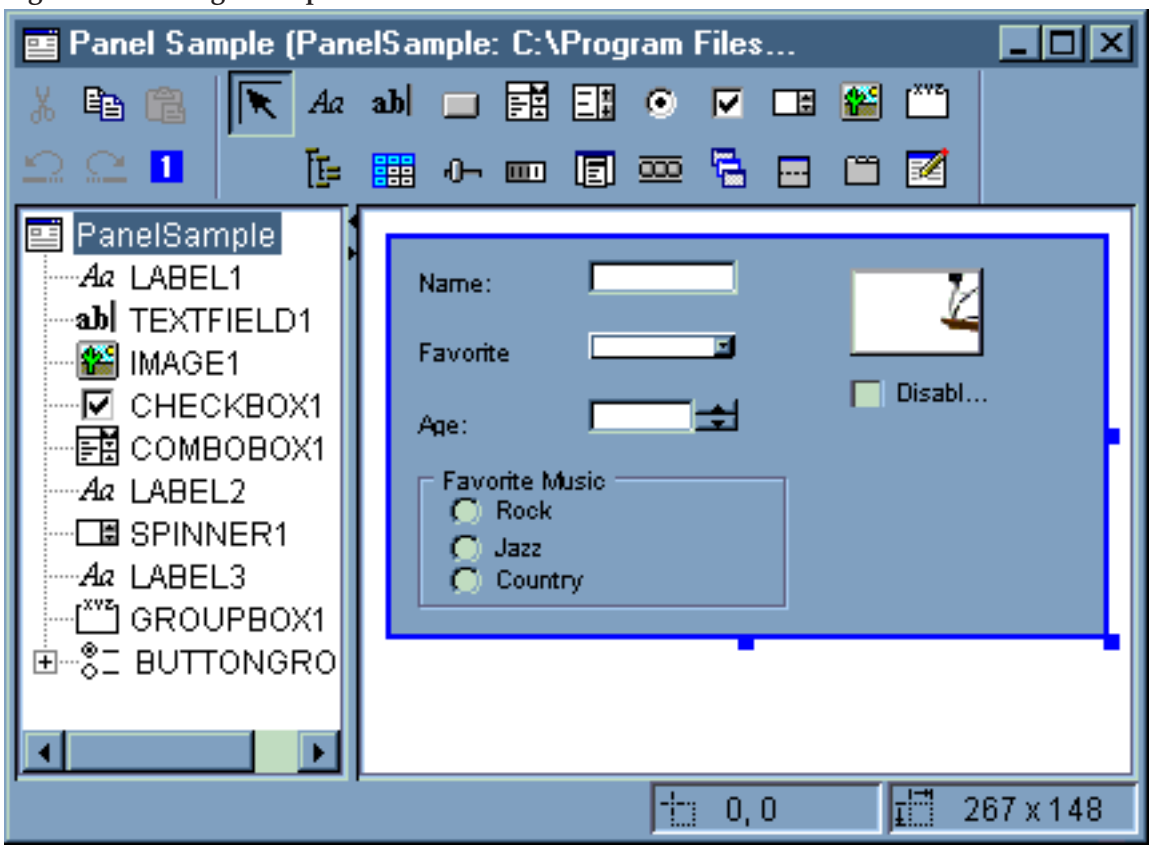
Creating a panel with GUI Builder

Creating a panel with GUI Builder is simple. From the menu bar on the main GUI Builder window, select **File** → **New File**.

From the menu bar on the GUI Builder **File** window, click the Insert New Panel icon  to display a panel builder where you can insert the components for your panel. The toolbar buttons on the **Panel** window represent various components that you can add to the panel. Select the component you want and then click on the place you want to position it.

The following picture shows a panel that has been created with several of the options available to you.

Figure 1: Creating a sample Panel with GUI Builder



The sample panel in Figure 1 uses the following DataBean code to bring together the various components:

```
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";
    }
}
```

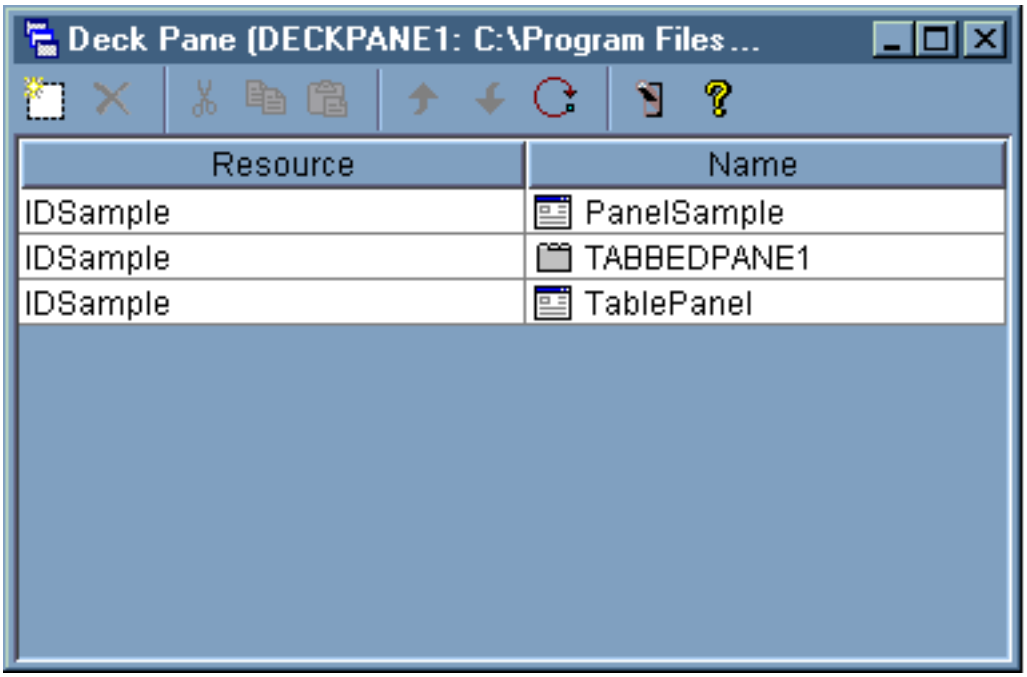
The panel is the most simple component available within the GUI Builder, but from a simple panel you can build great UI applications.

Creating a deck pane with GUI Builder

GUI Builder makes creating a deck pane simple. From the menu bar on the GUI Builder window, select **File** --> **New File**.

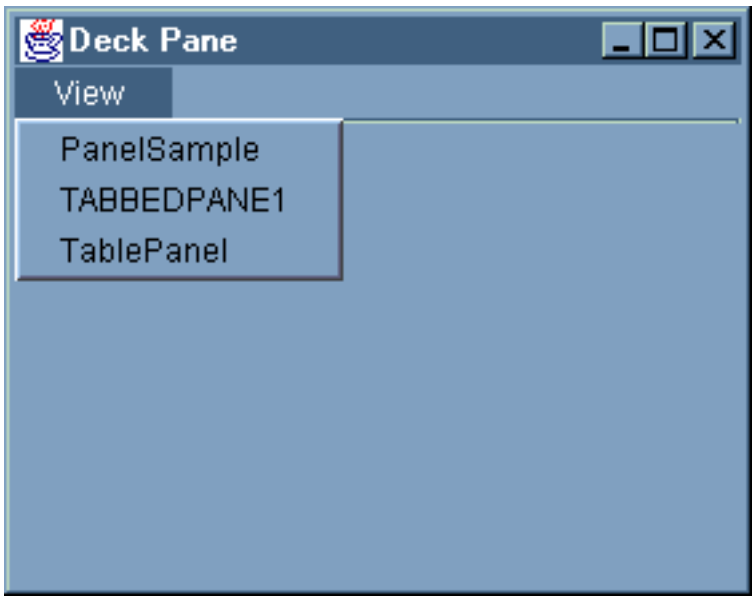
From the menu bar on the GUI Builder **File** window, click the **Insert Deck Pane** tool button  to display a panel builder where you can insert the components for your deck pane. In the following example, three components are added.

Figure 1: Creating a Deck Pane with GUI Builder



After you create the deck pane, click the **Preview** tool button  to preview it. A deck pane looks plain until you select the **View** menu.

Figure 2: Previewing the Deck Pane with GUI Builder



From the Deck Pane **View** menu, select the component you want to view. For this example, you can choose to view the PanelSample, TABBEDPANE1, or the TablePanel. The following figures illustrate what you see when you view these components.

Figure 3: Viewing the PanelSample with GUI Builder

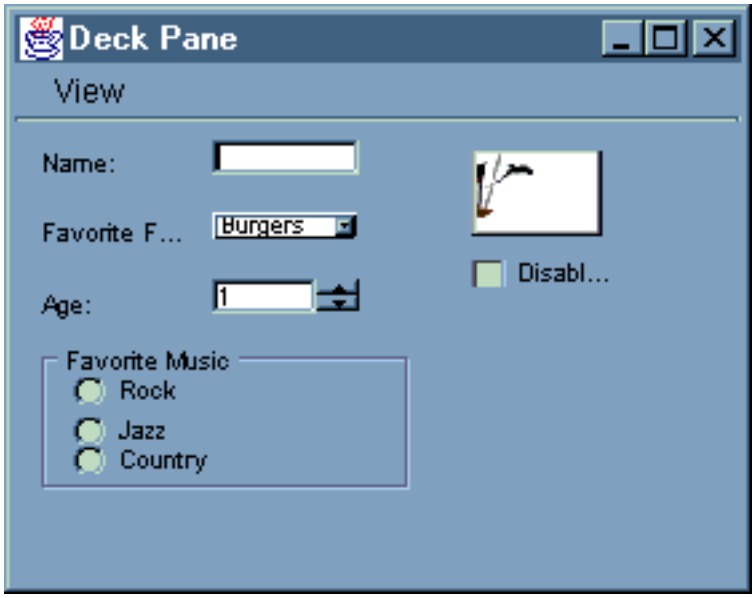


Figure 4: Viewing TABBEDPANE1 with GUI Builder

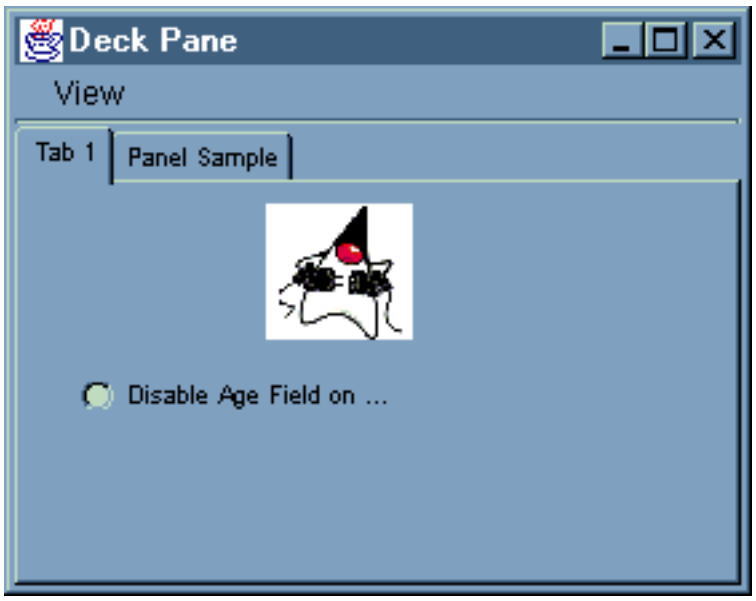
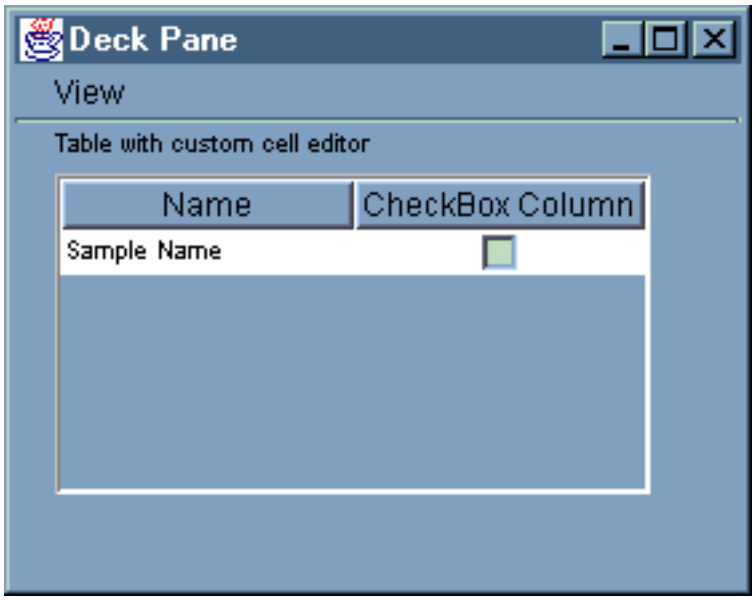


Figure 5: Viewing the TablePanel with GUI Builder

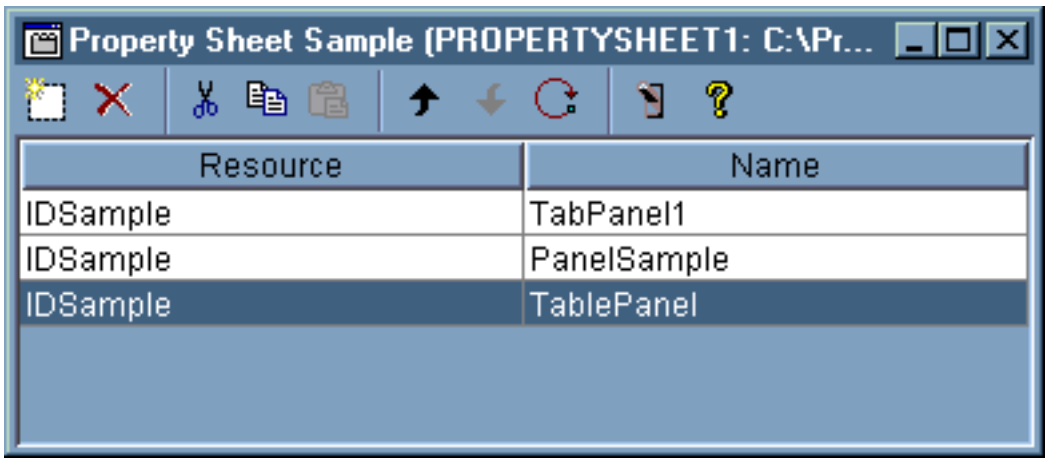


Creating a property sheet with GUI Builder

GUI Builder makes creating a property sheet simple. From the menu bar on the main GUI Builder window, select **File** --> **New File**.

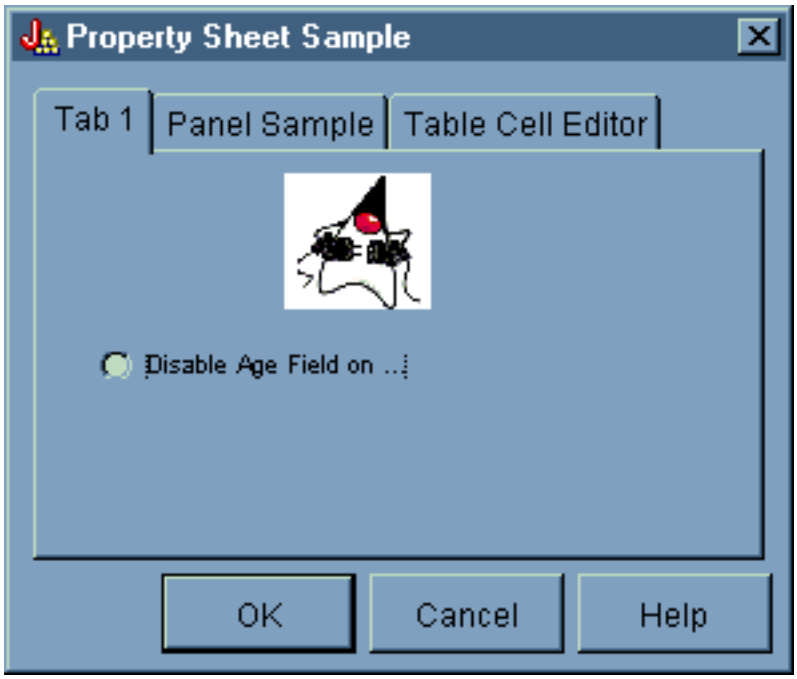
From the menu bar on the GUI Builder **File** window, click the Insert Property Sheet icon  to display a panel builder where you can insert the components for your property sheet.

Figure 1: Creating a Property Sheet with GUI Builder



After you create the property sheet, use the  icon to preview it. For this example, you can choose from three tabs.

Figure 2: Previewing a Property Sheet with GUI Builder

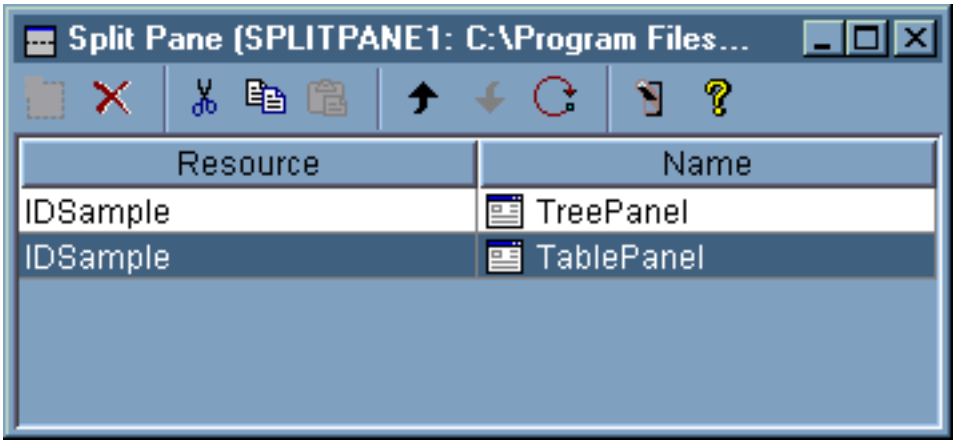


Creating a split pane with GUI Builder

GUI Builder makes creating a split pane simple. From the menu bar on the main GUI Builder window, select **File** --> **New File**.

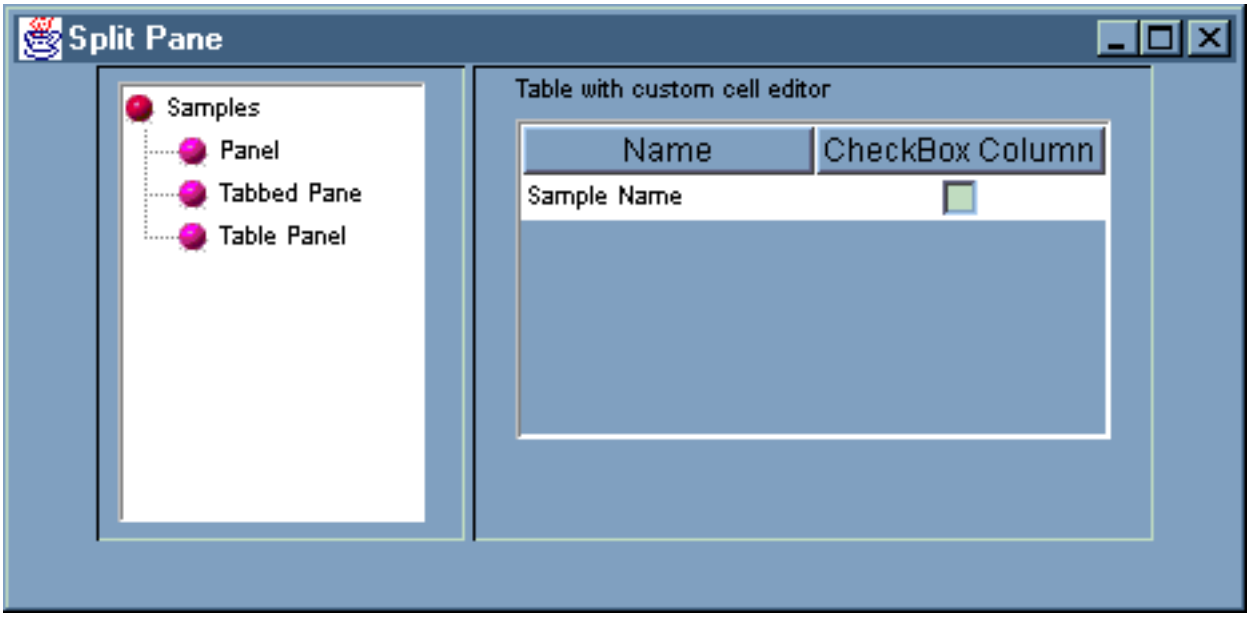
From the menu bar on the GUI Builder **File** window, click the Insert Split Pane tool button  to display a panel builder where you can insert the components you want in your split pane. In the following example, two components are added.

Figure 1: Creating a Split Pane with GUI Builder



After you create the split pane, click the **Preview** tool button  icon to preview it, as shown in Figure 2.

Figure 2: Previewing the Split Pane with GUI Builder

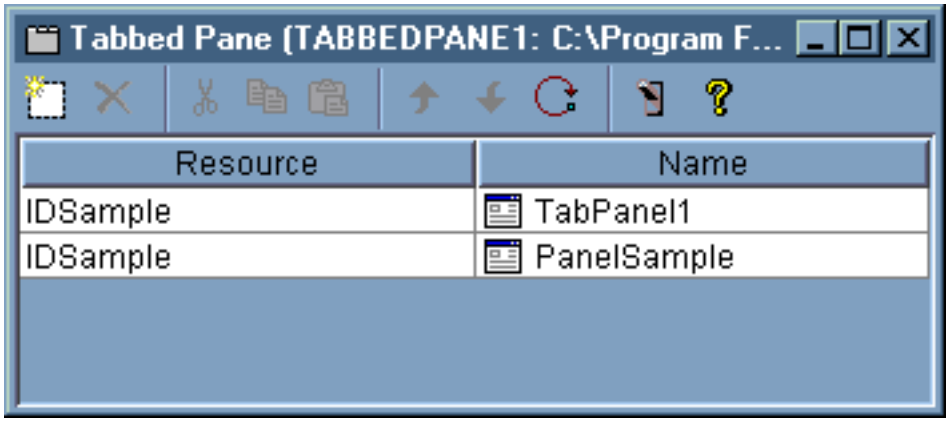


Creating a tabbed pane with GUI Builder

GUI Builder makes creating a tabbed pane simple. From the menu bar on the main GUI Builder window, select **File** --> **New File**.

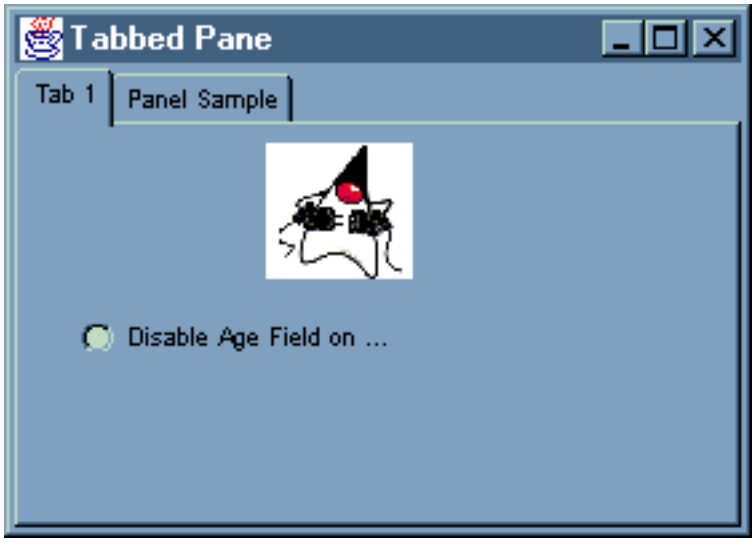
From the menu bar on the GUI Builder **File** window, click the Insert Tabbed Pane icon  to display a panel builder where you can insert the components for your tabbed pane. In the following example, two components are added.

Figure 1: Creating a Tabbed Pane in GUI Builder



After you create the tabbed pane, click the **Preview** tool button  to preview it.

Figure 2: Previewing the Tabbed Pane with GUI Builder



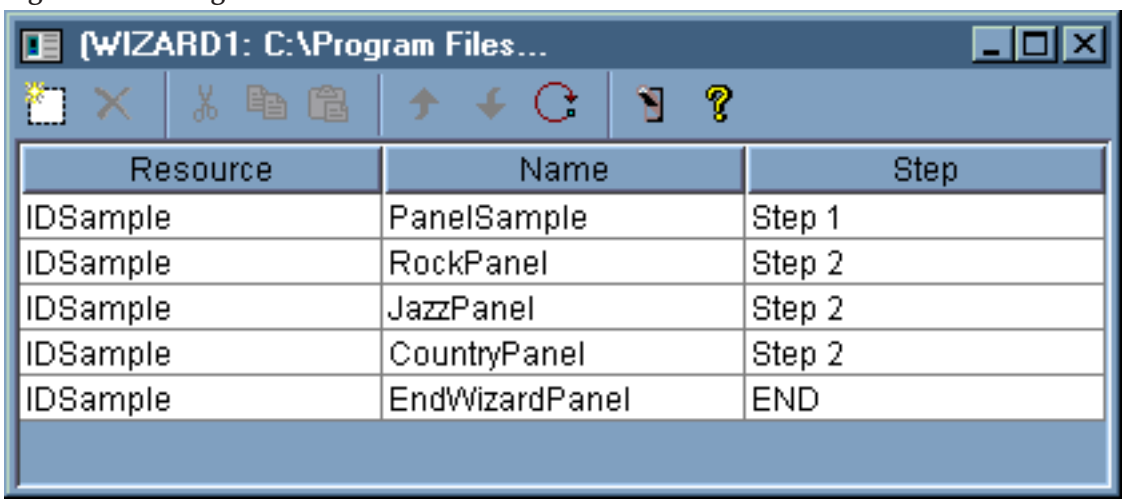
Creating a wizard with GUI Builder

GUI Builder makes creating a wizard interface simple. From the menu bar on the GUI Builder window, select **File** --> **New File**.

From the menu bar on the GUI Builder **File** window, click the Insert Wizard toobar button

to display a panel builder where you can add panels to the wizard.

Figure 1: Creating a Wizard with GUI Builder



After you have create the wizard, use the **Preview** tool button

to preview it. Figure 2 shows the panel that first displays for this example.

Figure 2: Previewing the first wizard panel with GUI Builder

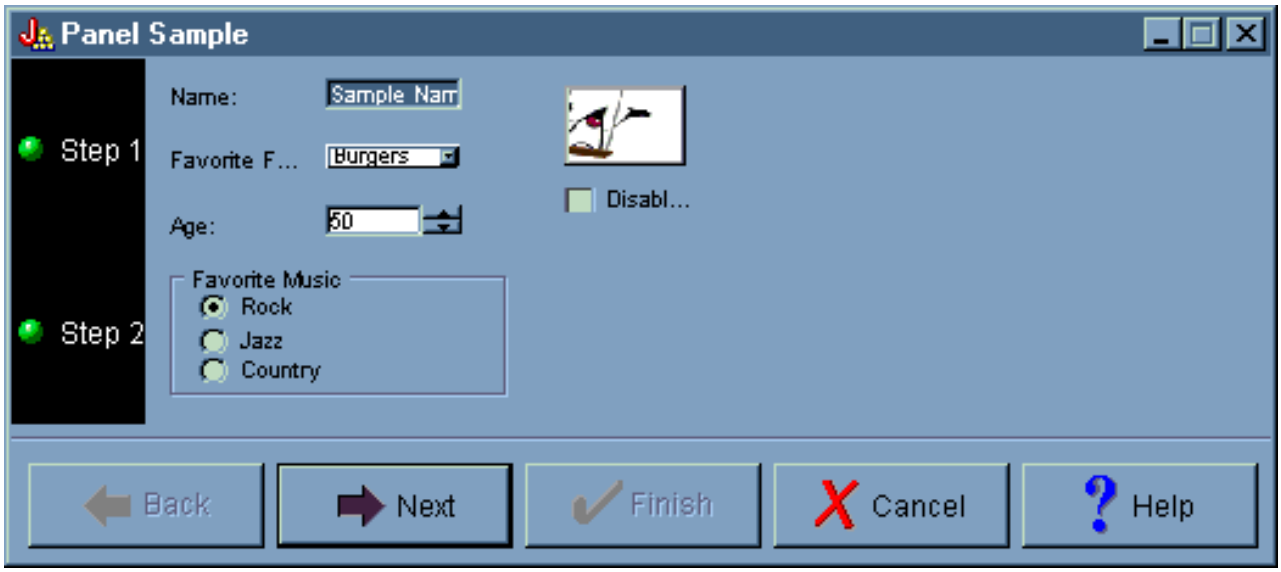
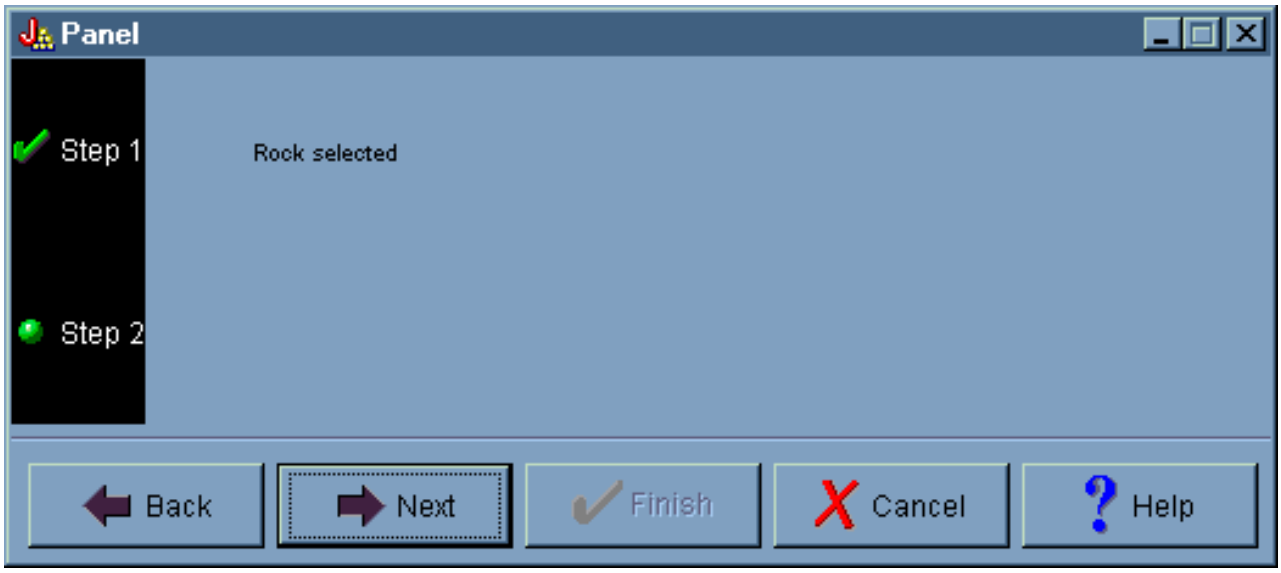


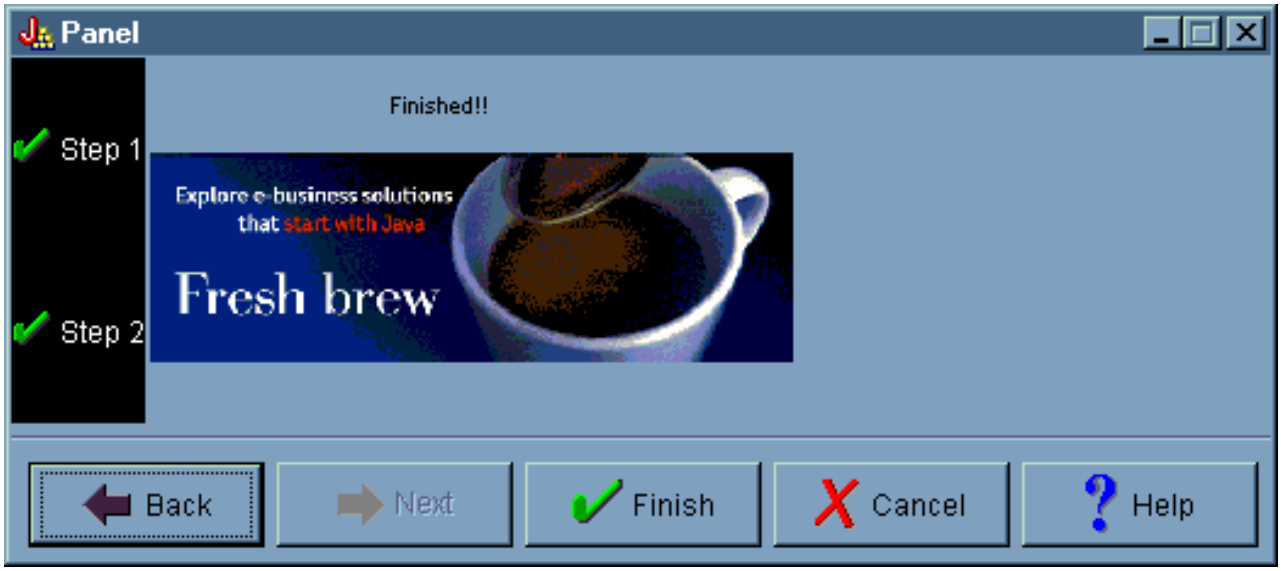
Figure 2 shows the second panel that displays when the user selects **Rock** and clicks **Next**.

Figure 3: Previewing the second wizard panel with GUI Builder



Clicking **Next** on the second wizard panel displays the final wizard panel, as shown in Figure 4.

Figure 4: Previewing the final wizard panel with GUI Builder

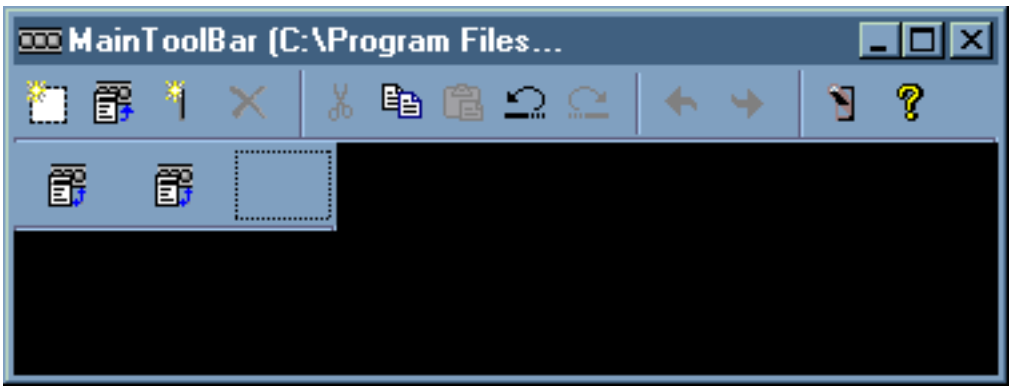


Creating a toolbar with GUI Builder

GUI Builder makes creating a toolbar simple. From the menu bar on the GUI Builder window, select **File --> New File**.

From the menu bar on the GUI Builder **File** window, click the **Insert Tool Bar** tool button to display a panel builder where you can insert the components for your toolbar.

Figure 1: Creating a Tool Bar with GUI Builder



After you create the toolbar, click the **Preview** tool button  to preview it. For this example, you can use the toolbar to display either a property sheet or wizard.

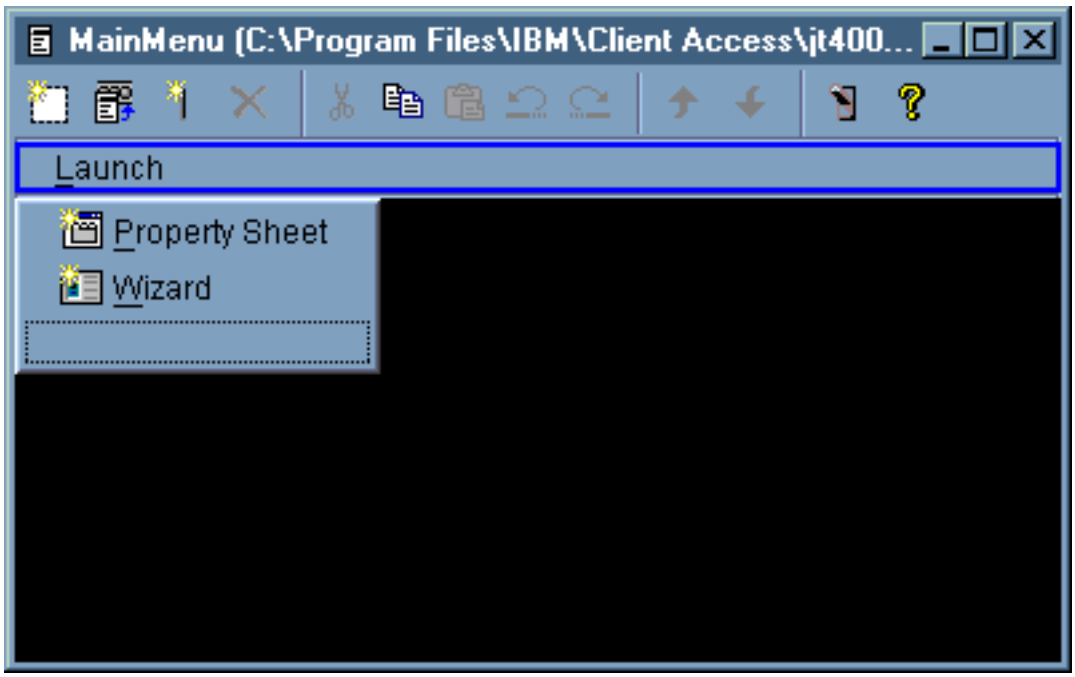
Figure 2: Previewing the Tool Bar with GUI Builder

Creating a menu bar with GUI Builder

GUI Builder makes creating a menu bar simple. From the menu bar on the GUI Builder window, select **File** --> **New File**.

From the tool bar on the GUI Builder **File** window, click the **Insert Menu** tool button to create a panel builder where you can insert the components for your menu.

Figure 1: GUI Builder: Creating a Menu



After you create the menu, use the **Preview** tool button  to preview it. For this example, from the newly created **Launch** menu you can select either **Property Sheet** or **Wizard**. The following figures illustrate what you see when you select these menu items.

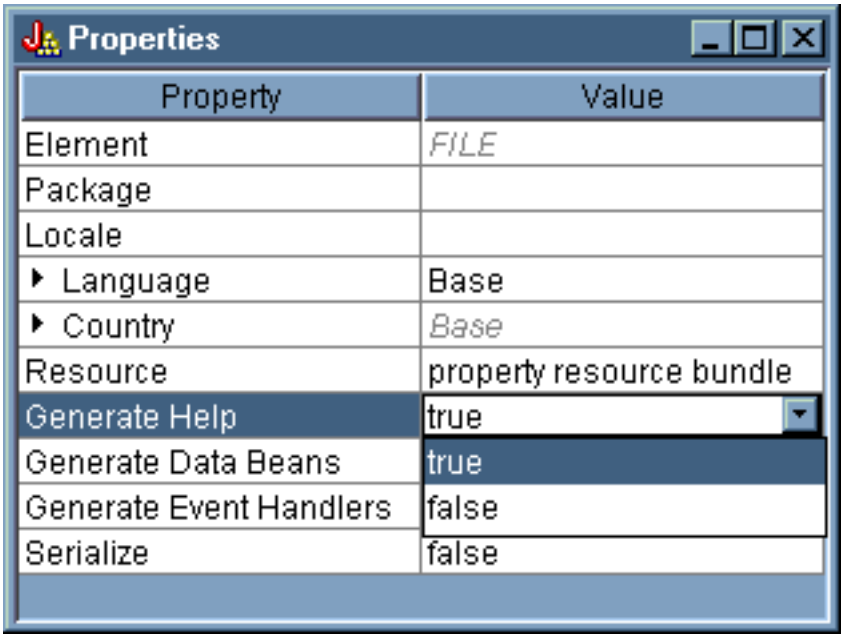
Figure 2: GUI Builder: Viewing Property Sheet on the Launch menu

Figure 3: GUI Builder: Viewing Wizard on the Launch menu

Example: Creating the Help Document

Creating help files with GUI Builder is simple. On the properties panel for the file you are working with, set ″Generate help″ to true.

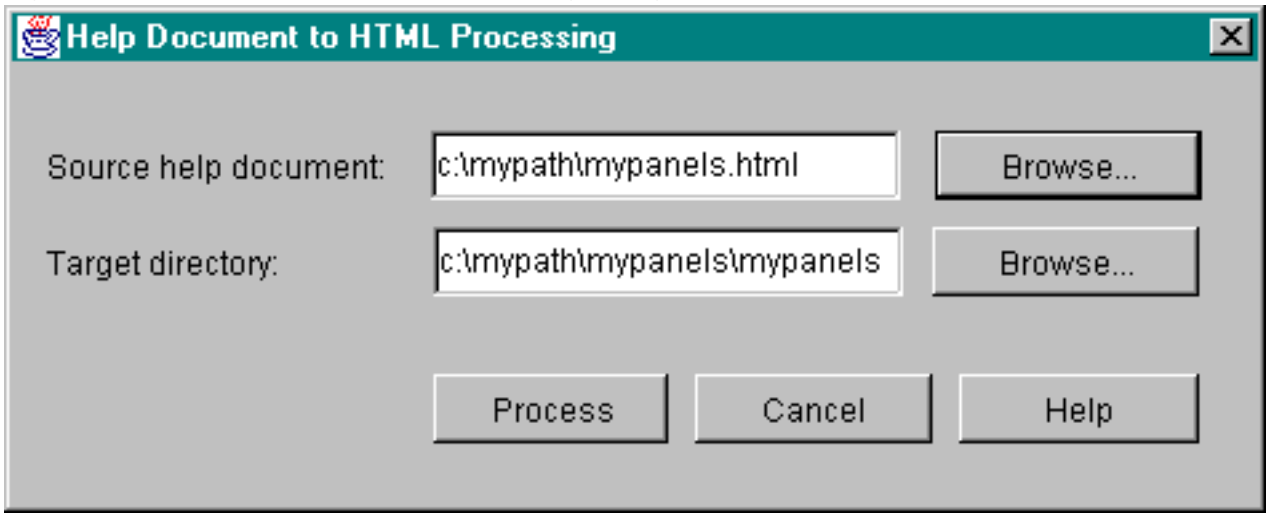
Figure 1: Setting the Generate Help property on the GUI Builder Properties panel



The GUI Builder creates an HTML framework called a Help Document, which you can edit.

In order to be used at runtime, the topics within the PDML file need to be separated into individual HTML files. When you run **Help Document to HTML Processing**, the topics are broken into individual files and put into a subdirectory named after the Help Document and PDML file. The runtime environment expects the individual HTML files to be in a subdirectory with the same name as the Help Document and PDML file. The **Help Document to HTML Processing** dialog gathers the information needed and calls the HelpDocSplitter program to do the processing:

Figure 2: Help Document to HTML Processing dialog



The Help Document to HTML Processing is started from a command prompt by typing:

```
jre com.ibm.as400.ui.tools.hdoc2htmViewer
```

Running this command requires that your classpath be set up correctly.

To use the Help Document to HTML Processing, you first select the Help Document that has the same name as the PDML file. Next, you specify a subdirectory with the same name as the Help Document and PDML file for output. Select ″Process″ to complete the processing.

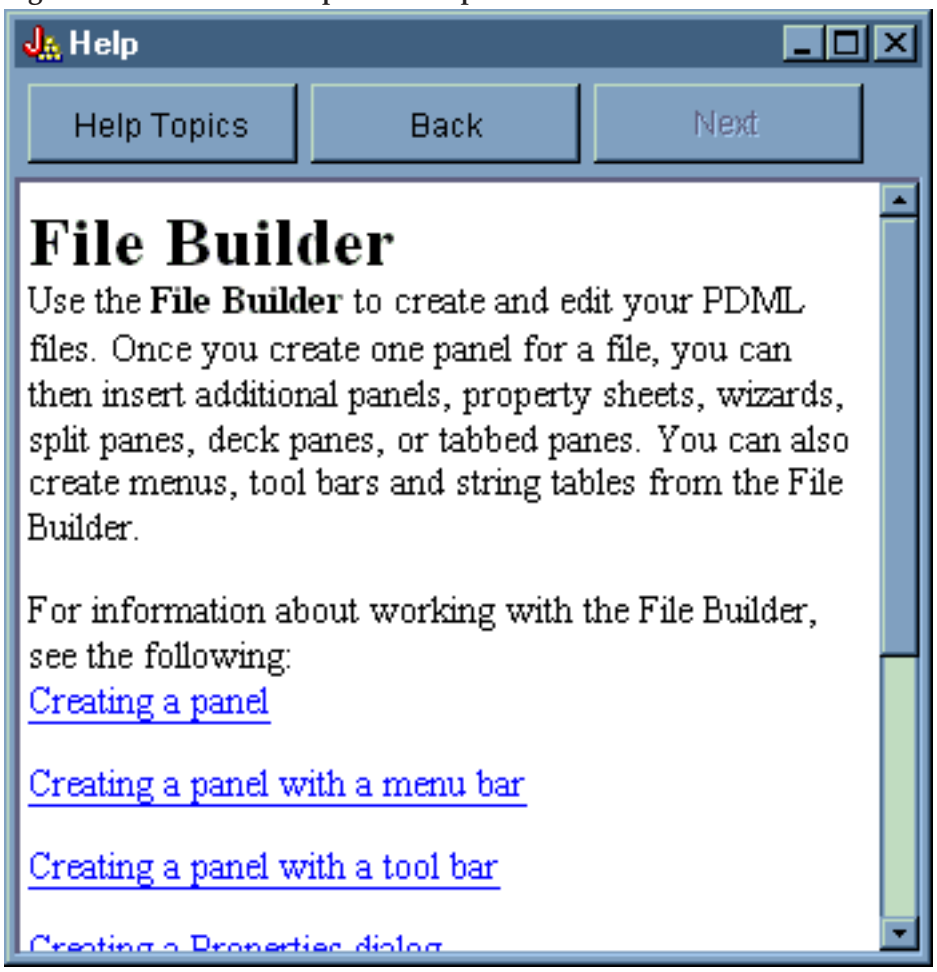
You can split up the help document from the command line with the following command:

```
jre com.ibm.as400.ui.tools.HelpDocSplitter "helpdocument.htm" [output directory]
```

This command runs the processing that breaks up the file. You provide the name of the Help Document as input along with an optional output directory. By default, a subdirectory of the same name as the Help Document is created and the resulting files are put in that directory.

This is an example of what a help file may look like:

Figure 3: GUI Builder help file example

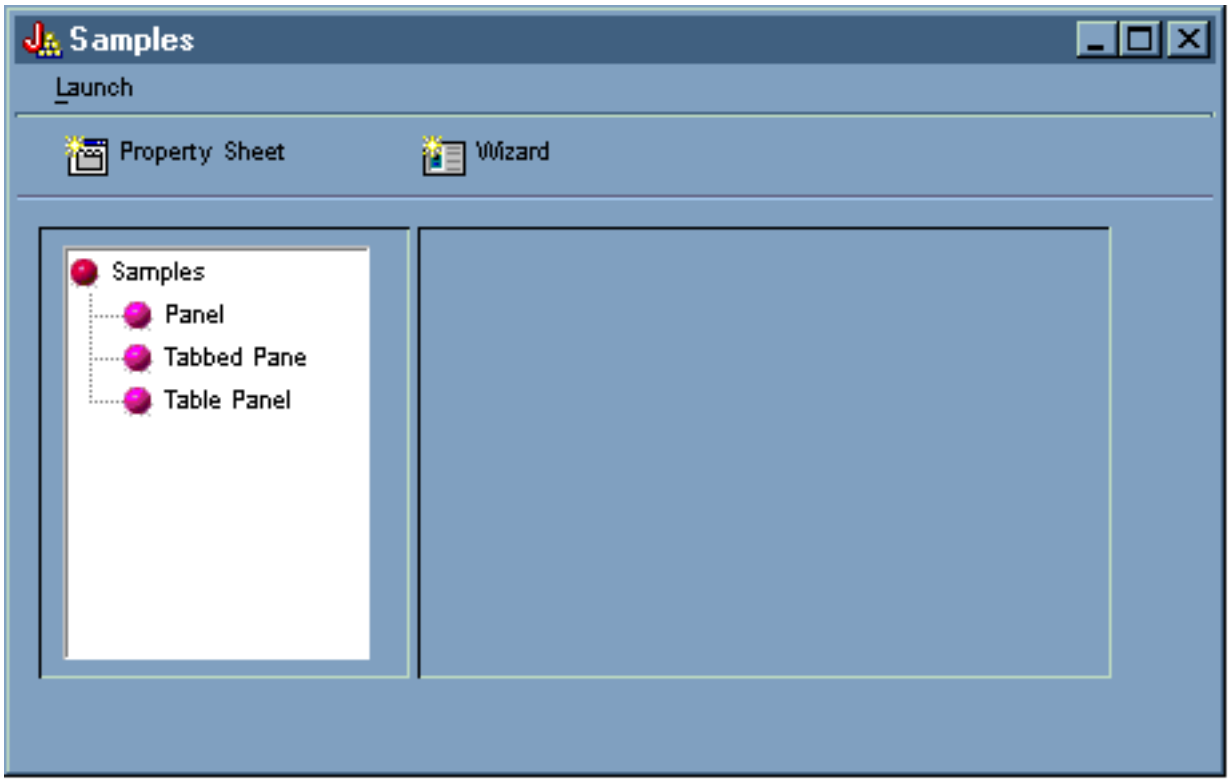


Example: Using GUI Builder

When the examples contained in this section are put together with the correct data beans working behind the scenes, you get a total GUI application.

Figure 1 shows the first panel that displays when you run this example.

Figure 1: The GUI Builder example main window



Notice that this screen allows you to use the dynamic panel manager. Figures 2 and 3 show how you can resize the window to be larger or smaller.

Figure 2: Resizing the GUI Builder example main window (larger)

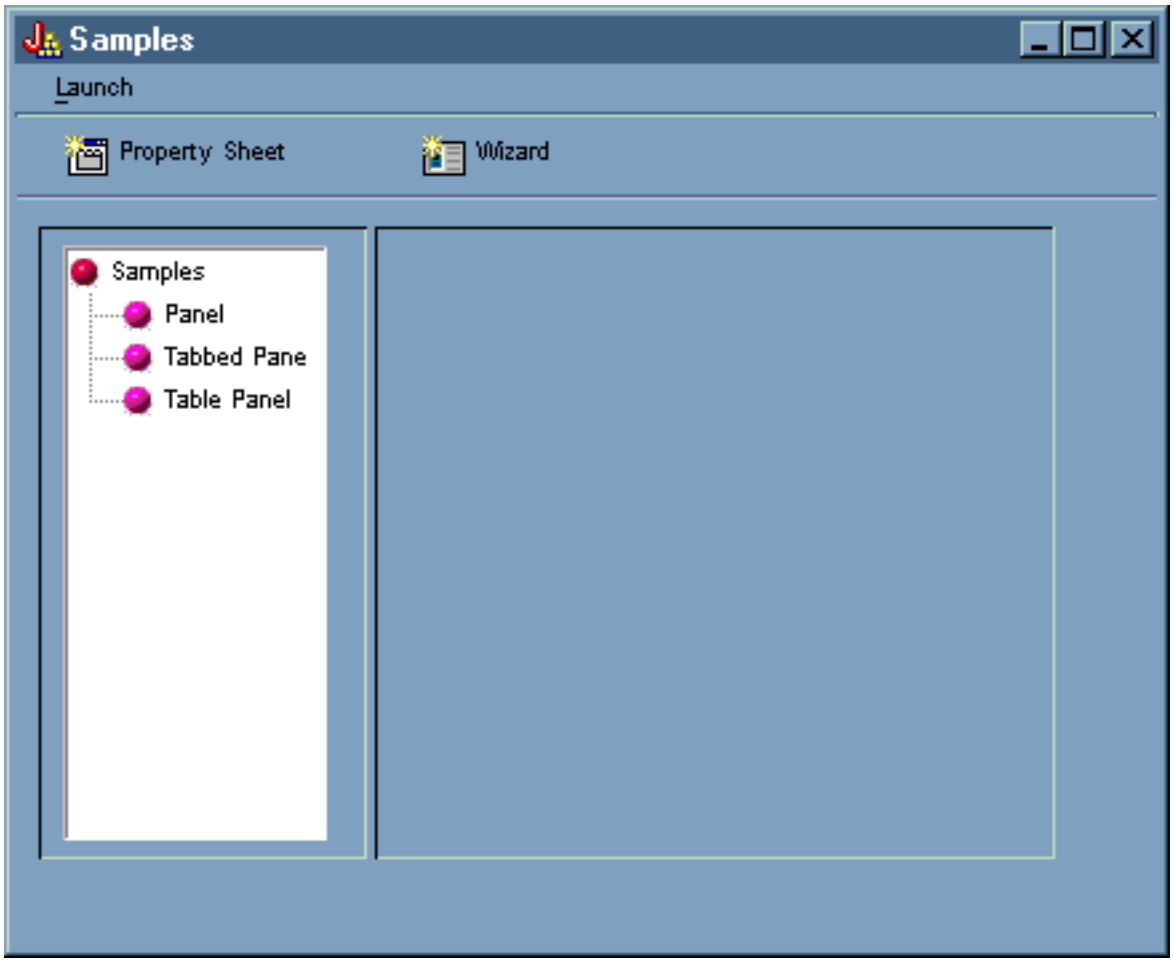
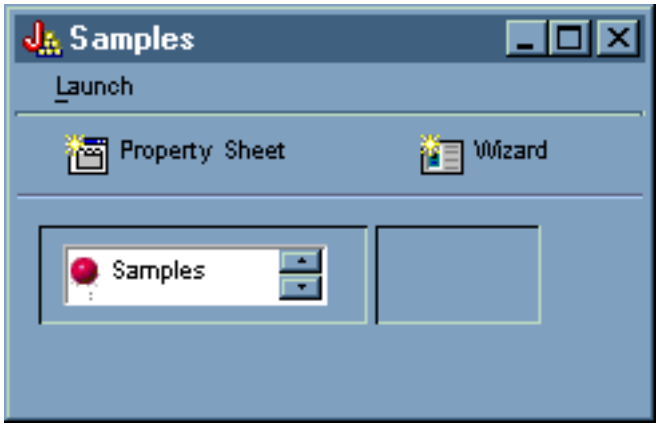


Figure 3: Resizing the GUI Builder example main window (smaller)



When you use the dynamic panel manager, while the size of the panel and the panel controls changes, the size of the text does not.

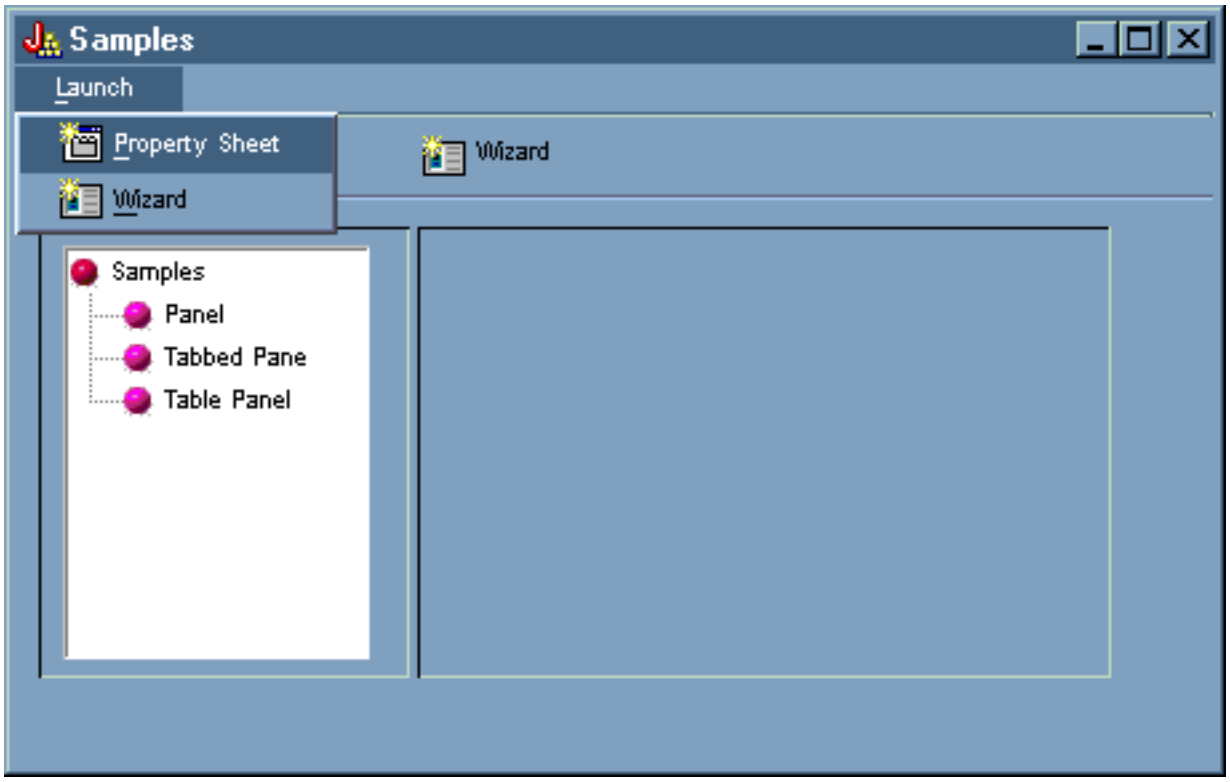
The panel allows you to perform the following actions:

- Launch a property sheet
- Launch a wizard
- Display the samples listed in the left pane

Launching the property sheet

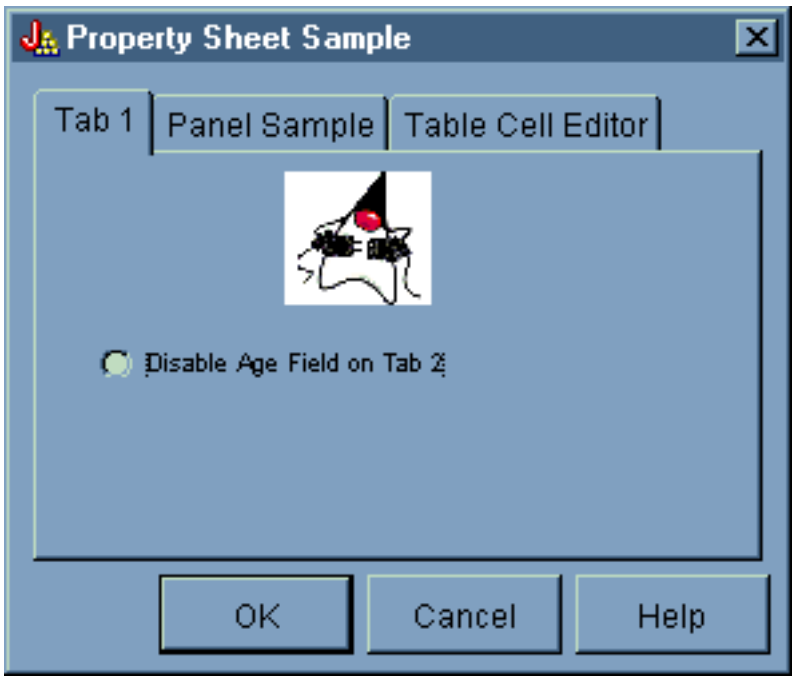
You can launch the property sheet by clicking the Property Sheet toolbar button or by using the **Launch** menu. Being able to choose between the toolbar and the menu illustrates linking menu items. Figure 4 shows **Property Sheet** being selected from **Launch** menu on the GUI Builder example main window.

Figure 4: Selecting Property Sheet from the Launch menu



Selecting **Property Sheet** displays the panel in Figure 5.

Figure 5: Property Sheet Sample dialog



When the Property Sheet Sample first appears, Tab 1 displays by default. Figures 6 and 7 show how the panel display changes when you select the other tabs.

Figure 6: Selecting the Panel Sample tab

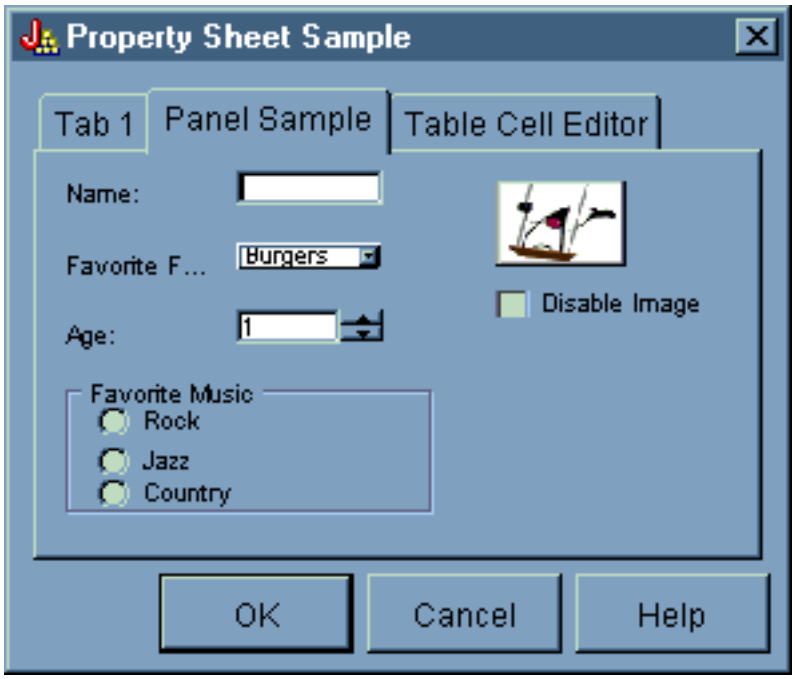
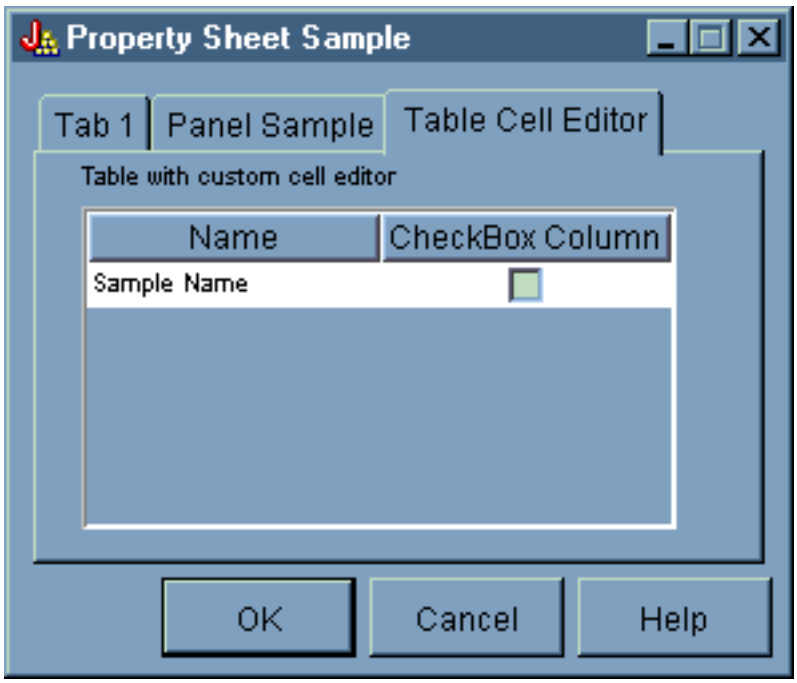


Figure 7: Selecting the Table Cell Editor tab



Launching the wizard

You can launch the wizard by clicking the Wizard toolbar button or by using the **Launch** menu. Being able to choose between the toolbar and the menu illustrates linking menu items. Figure 8 shows **Wizard** being selected from **Launch** menu on the GUI Builder example main window.

Figure 8: Selecting Wizard from the Launch menu

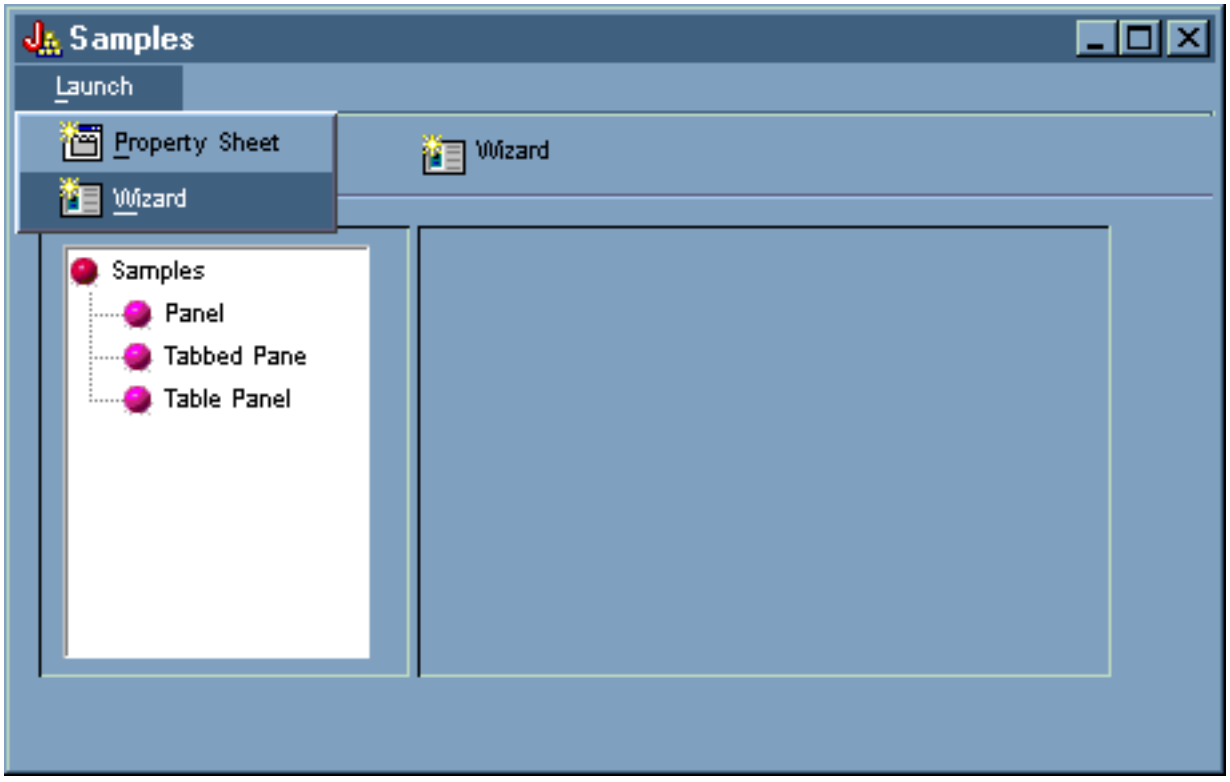
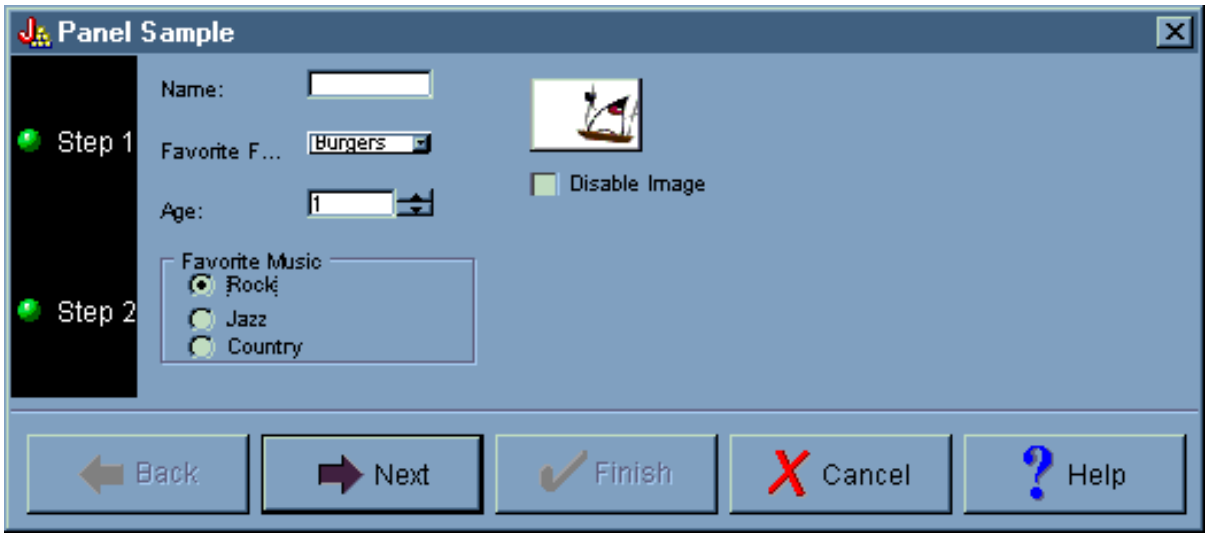


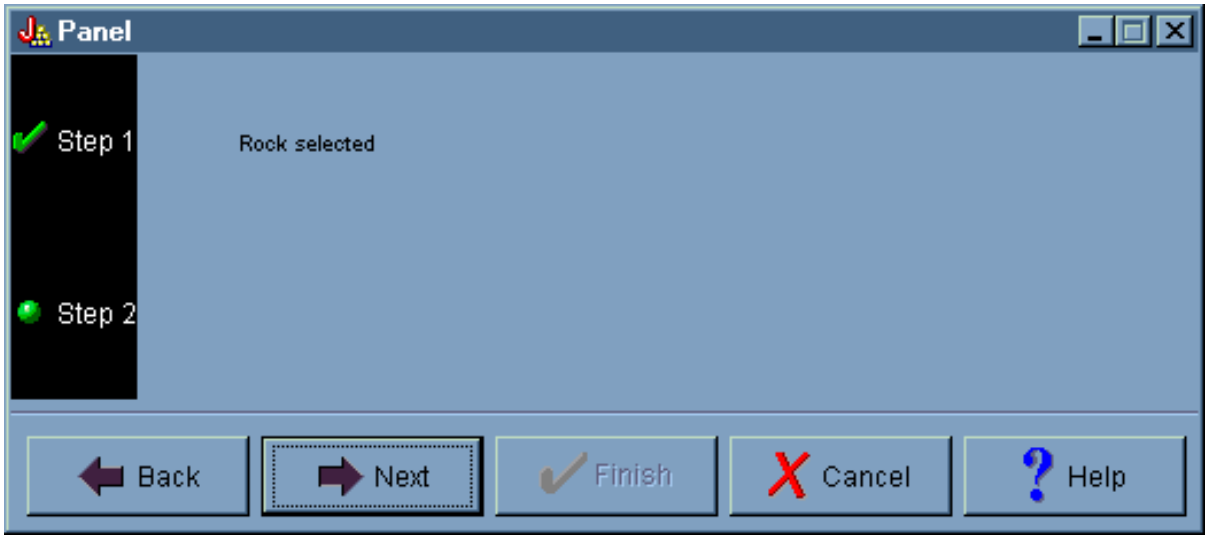
Figure 9 shows how the first wizard dialog gives you many options.

Figure 9: Selecting Rock in the first wizard dialog



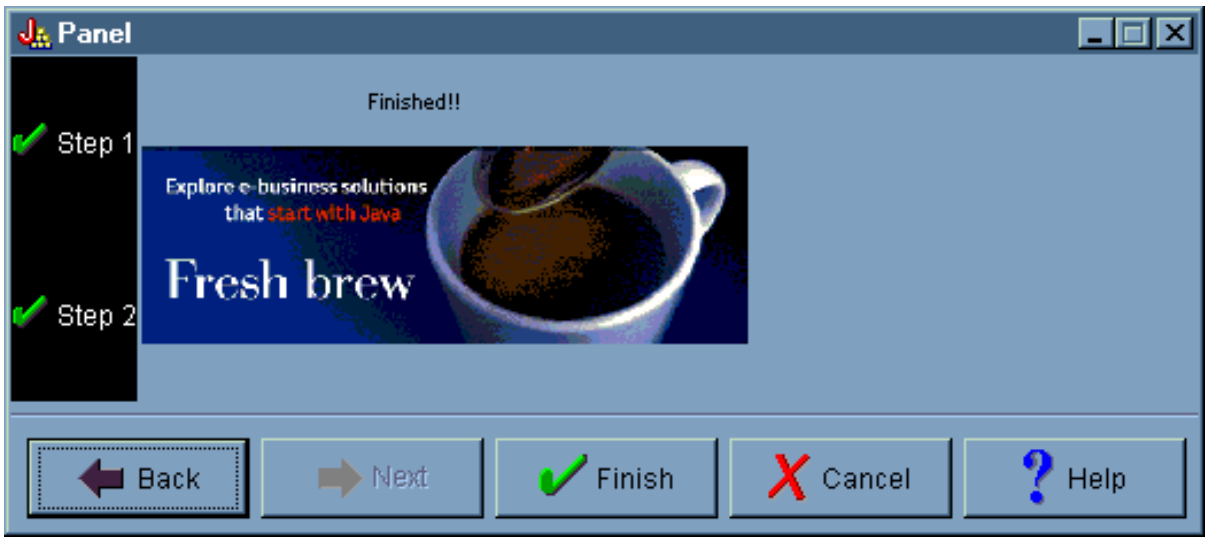
In the first wizard dialog, select **Rock** and click **Next** to display the second wizard dialog as shown in Figure 10.

Figure 10: The second wizard dialog (after selecting Rock)



On the second wizard dialog, click **Next** to display the final wizard dialog as shown in Figure 11.

Figure 11: The final wizard dialog



However, this example has been programmed to have a loop. Select **Country** in the first wizard dialog (Figure 12), then click **Next** to display the second wizard dialog (Figure 13). Clicking Next in the second wizard dialog loops back to display the first dialog again (Figure 14) instead of the final wizard dialog.

Figure 12: Selecting Country in the first wizard dialog

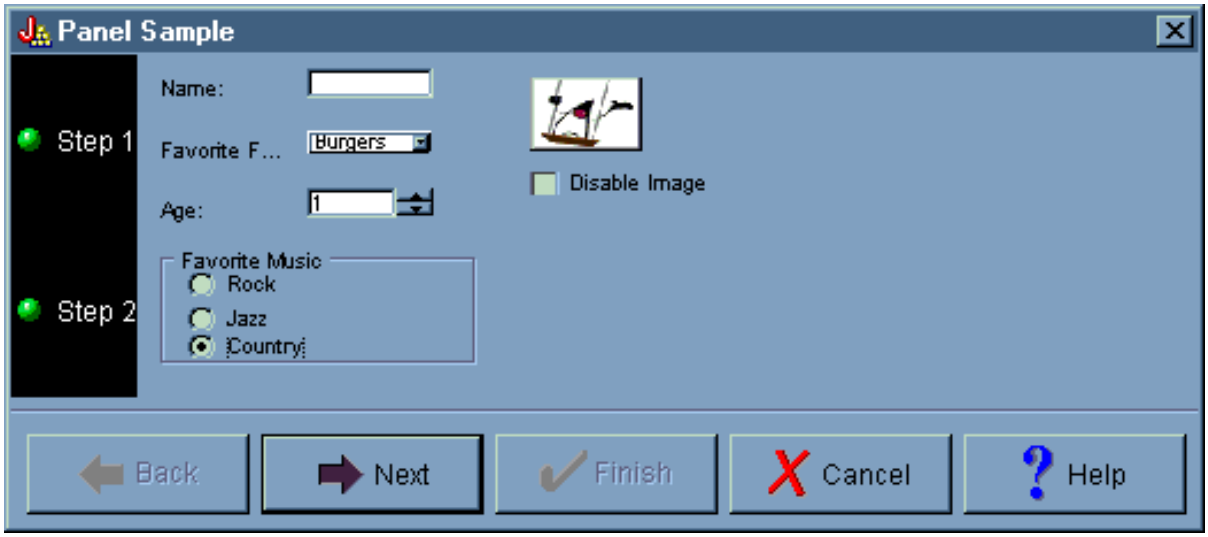


Figure 13: The second wizard dialog (after selecting Country)

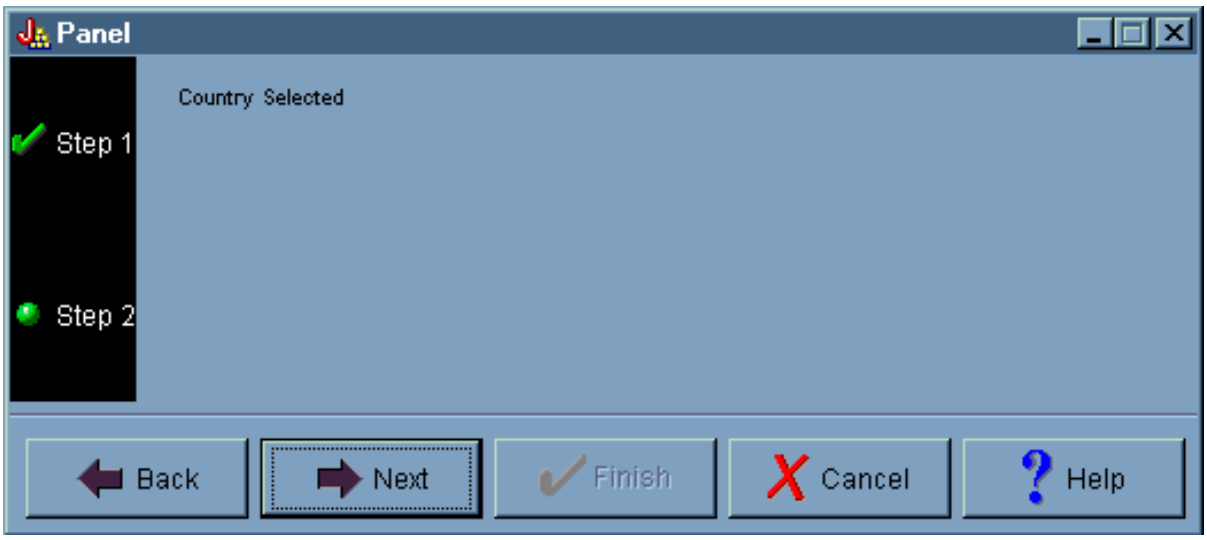
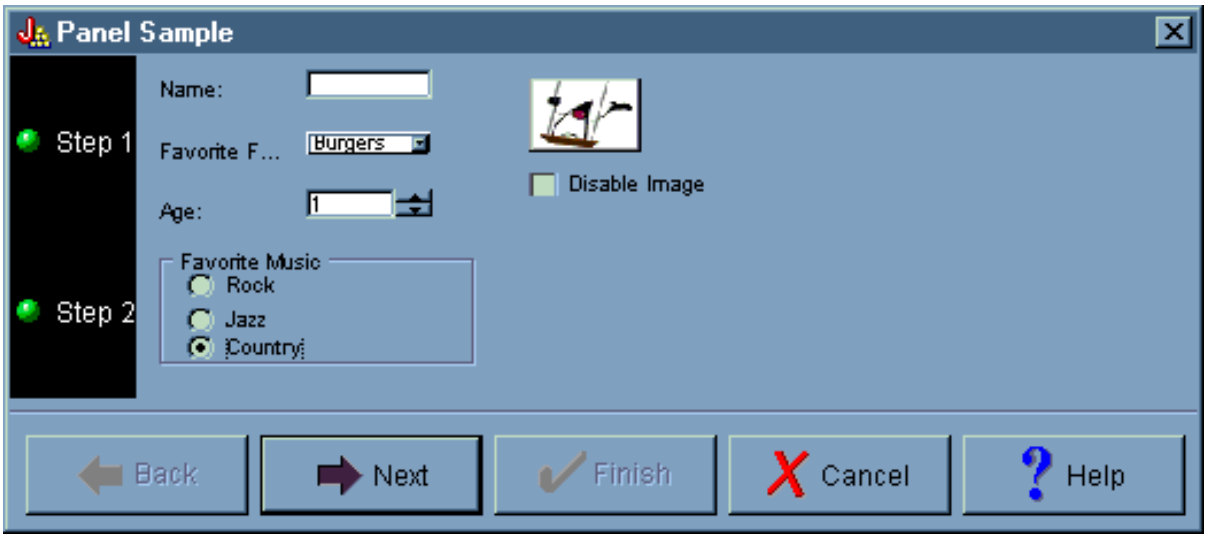


Figure 14: Looping back to the first wizard dialog

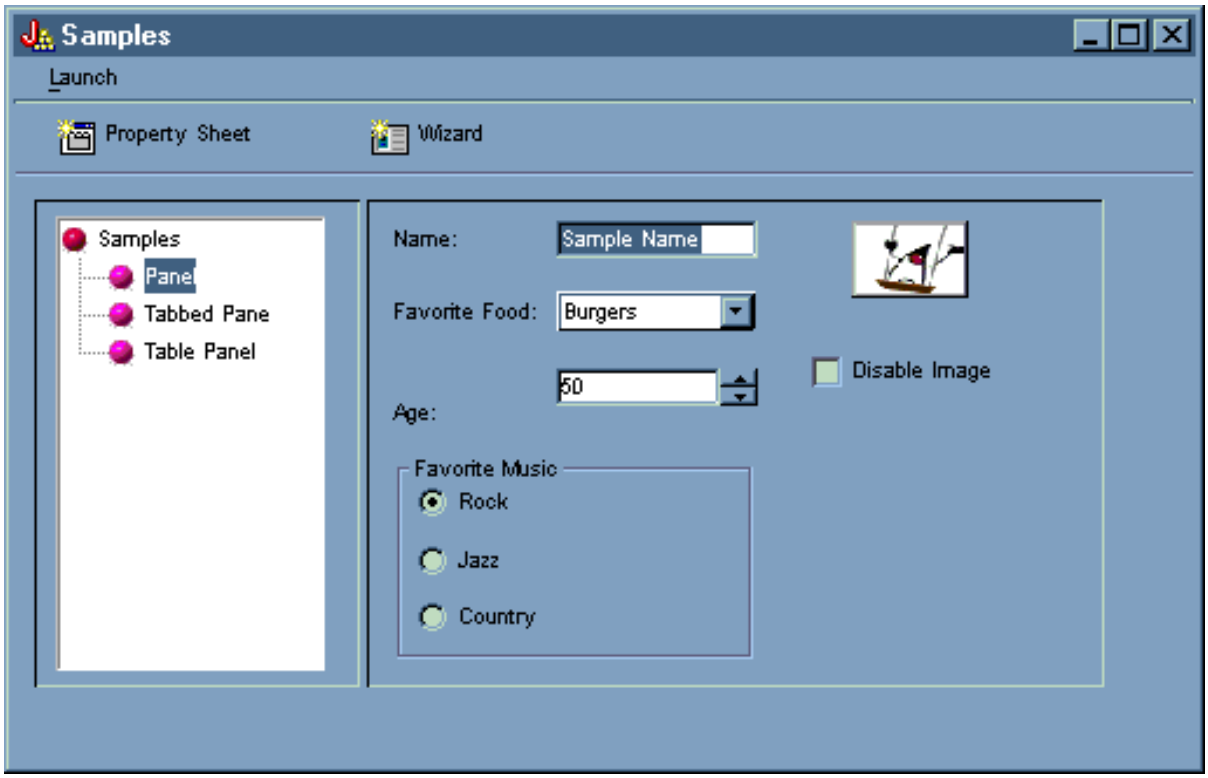


In other words, the programmer has determined that nobody can select country as their favorite form of music.

Displaying the samples

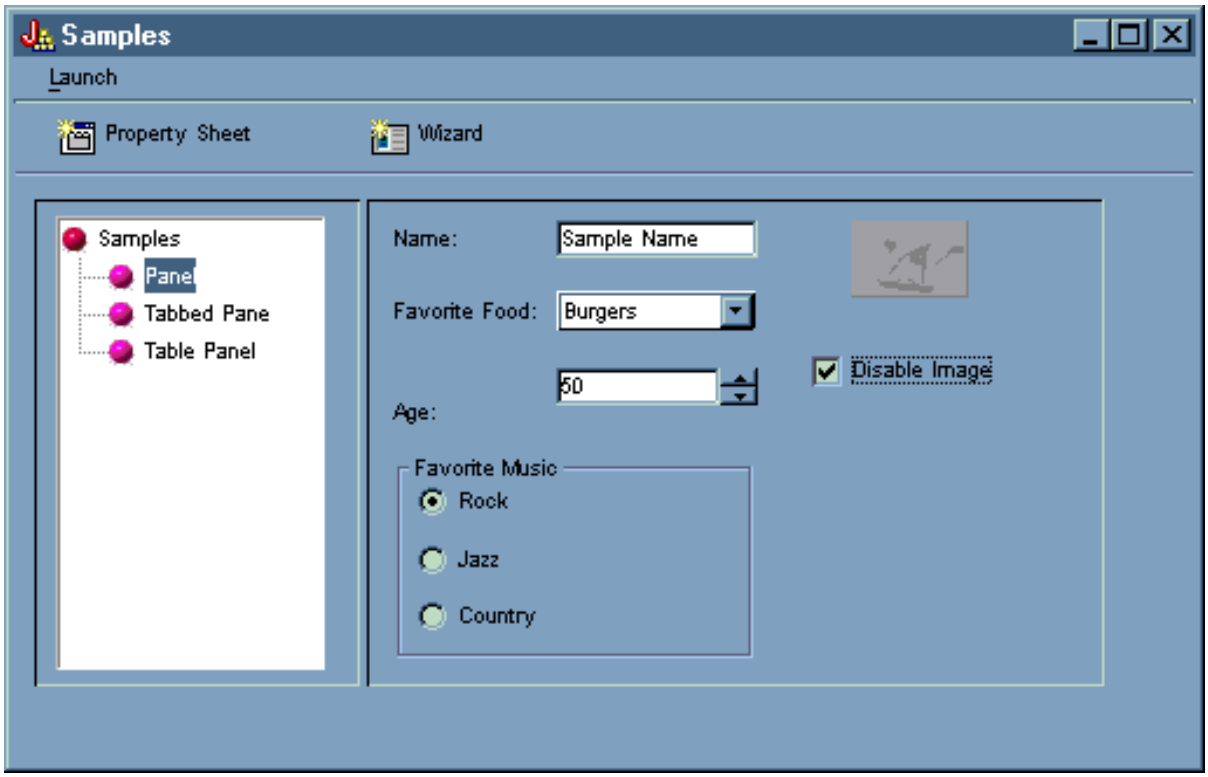
From the GUI Builder example main window, you can also select other functions from the left pane below the toolbar. Figure 15 shows how selecting **Panel** in the left pane displays the Panel sample in the right pane.

Figure 15: Selecting Panel in the left pane



The Panel sample has been programmed with an option to disable the image. Select **Disable Image** to display the same screen with the image shaded, as shown in Figure 16.

Figure 16: Selecting Disable Image in the right pane



The Panel sample also illustrates the drop-down list box option, as shown in Figure 17.

Figure 17: Selecting an item from the Favorite Food list in the right pane

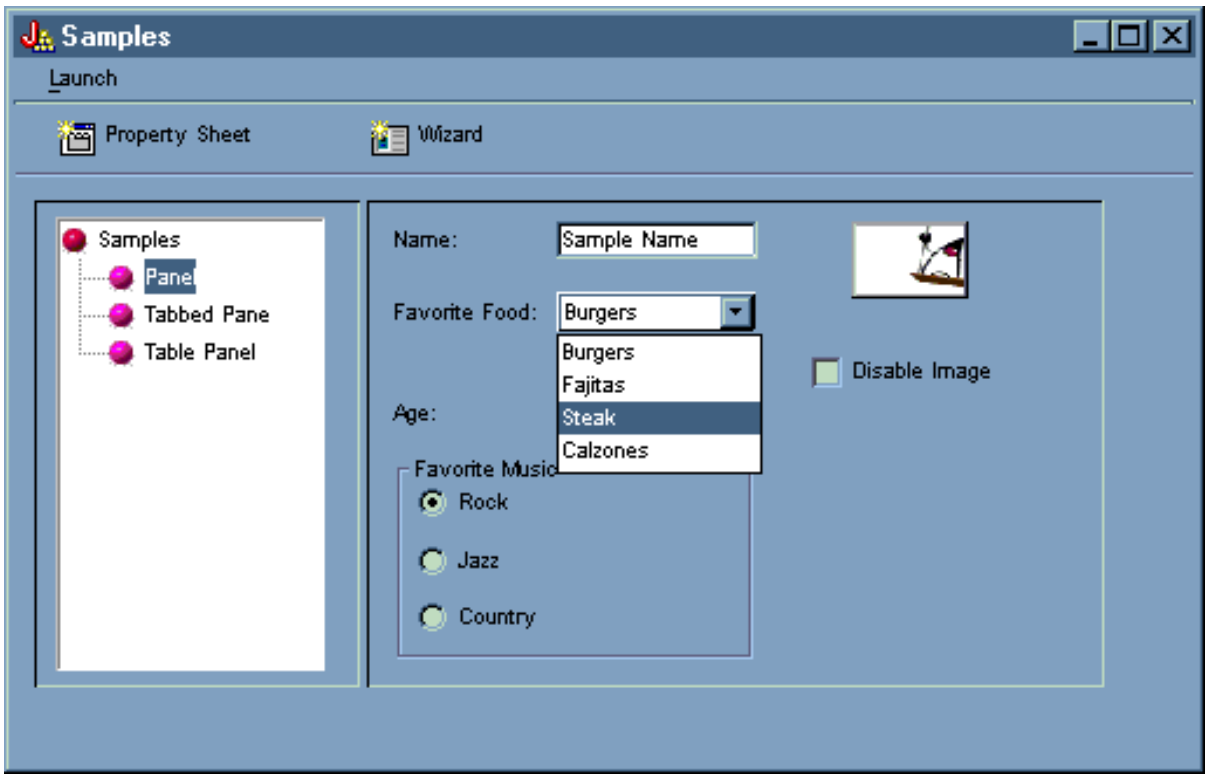


Figure 18 shows how selecting **Tabbed Pane** in the left pane of the GUI Builder example main window displays the Tabbed Pane sample in the right pane.

Figure 18: Selecting Tabbed Pane in the left pane

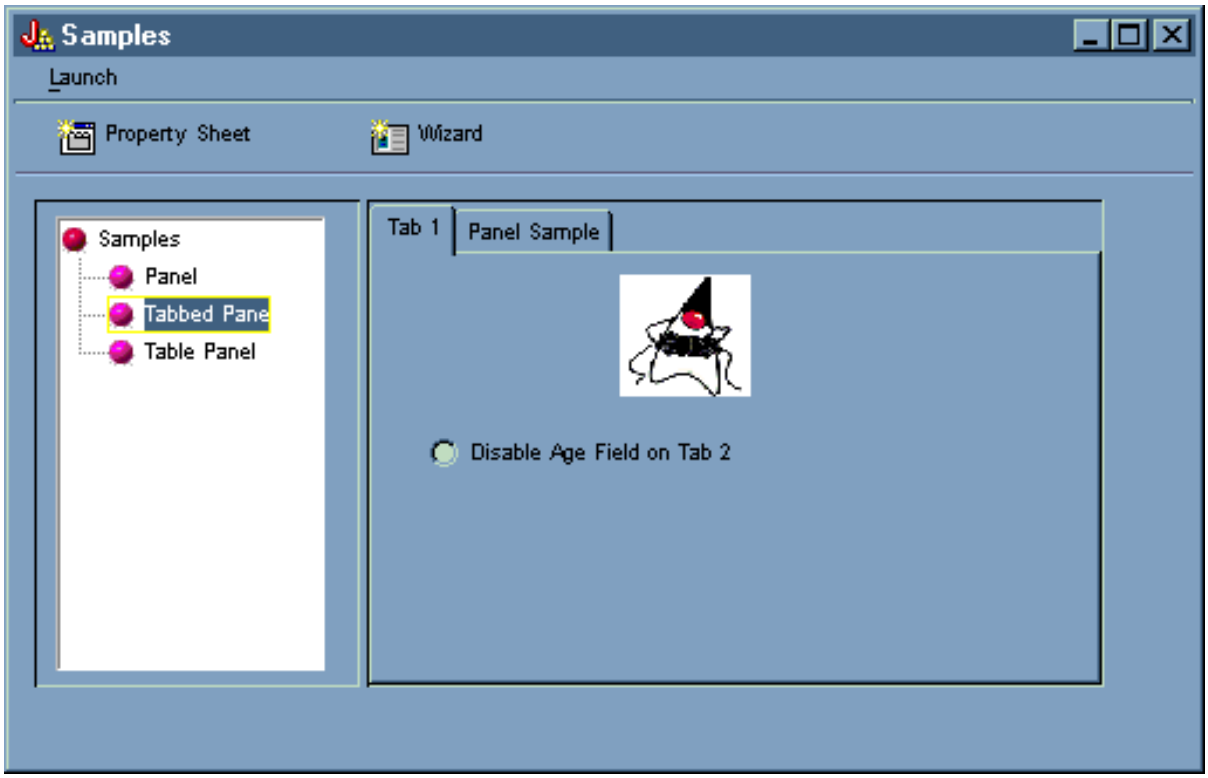
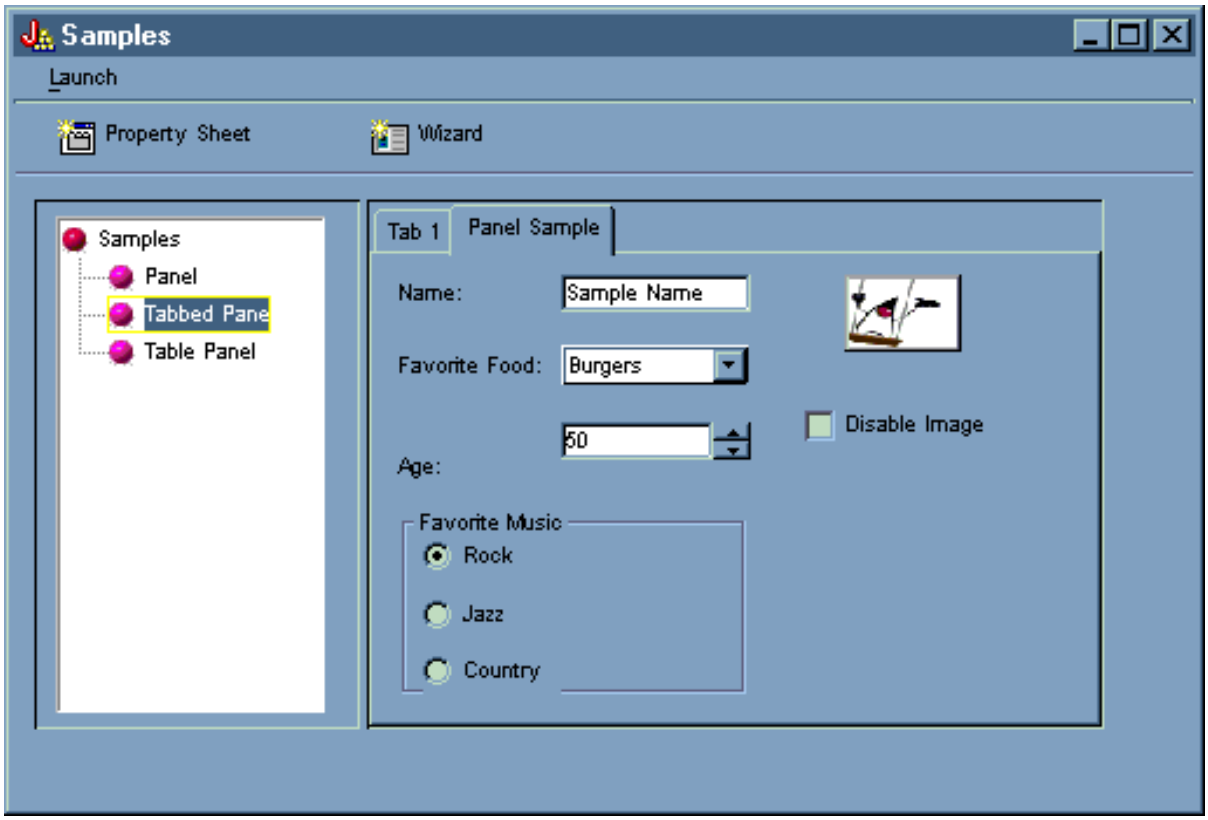


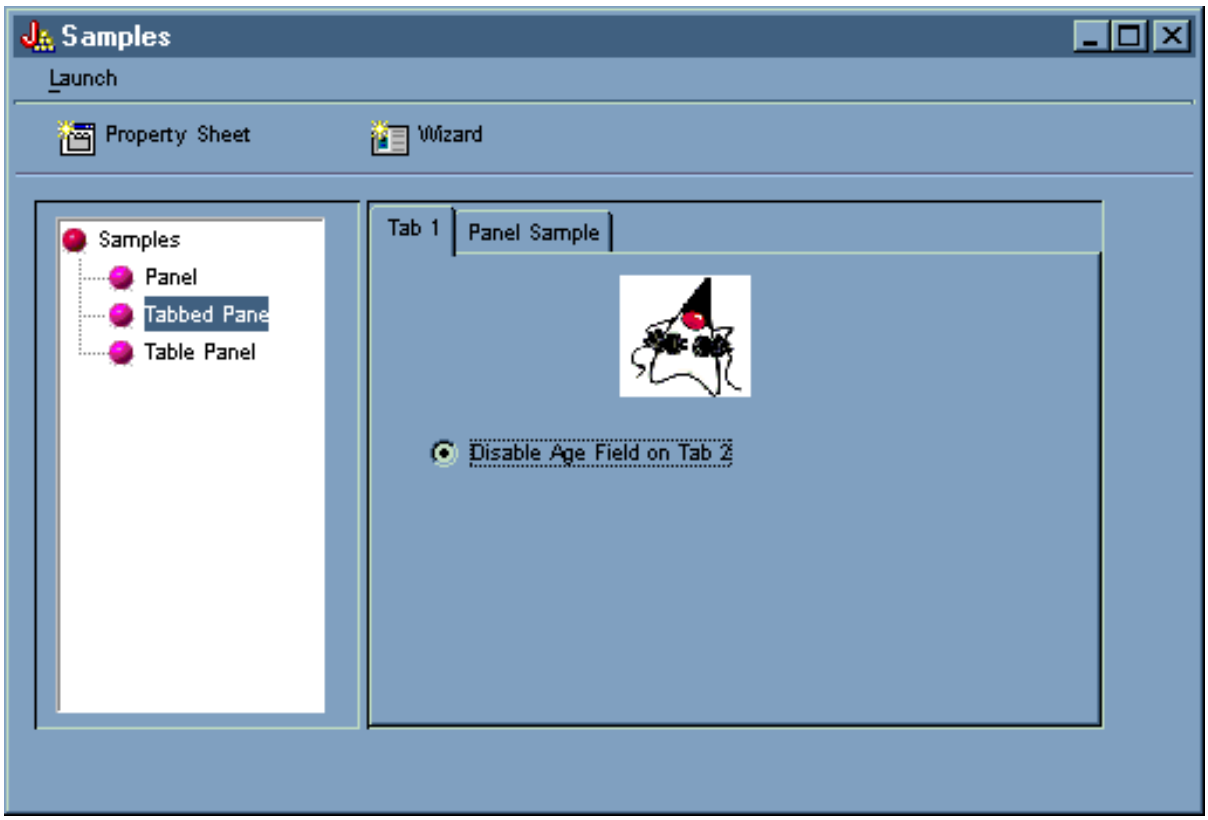
Figure 19 shows the results of selecting the **Panel Sample** tab in the right pane.

Figure 19: Selecting the Panel Sample tab in the right pane



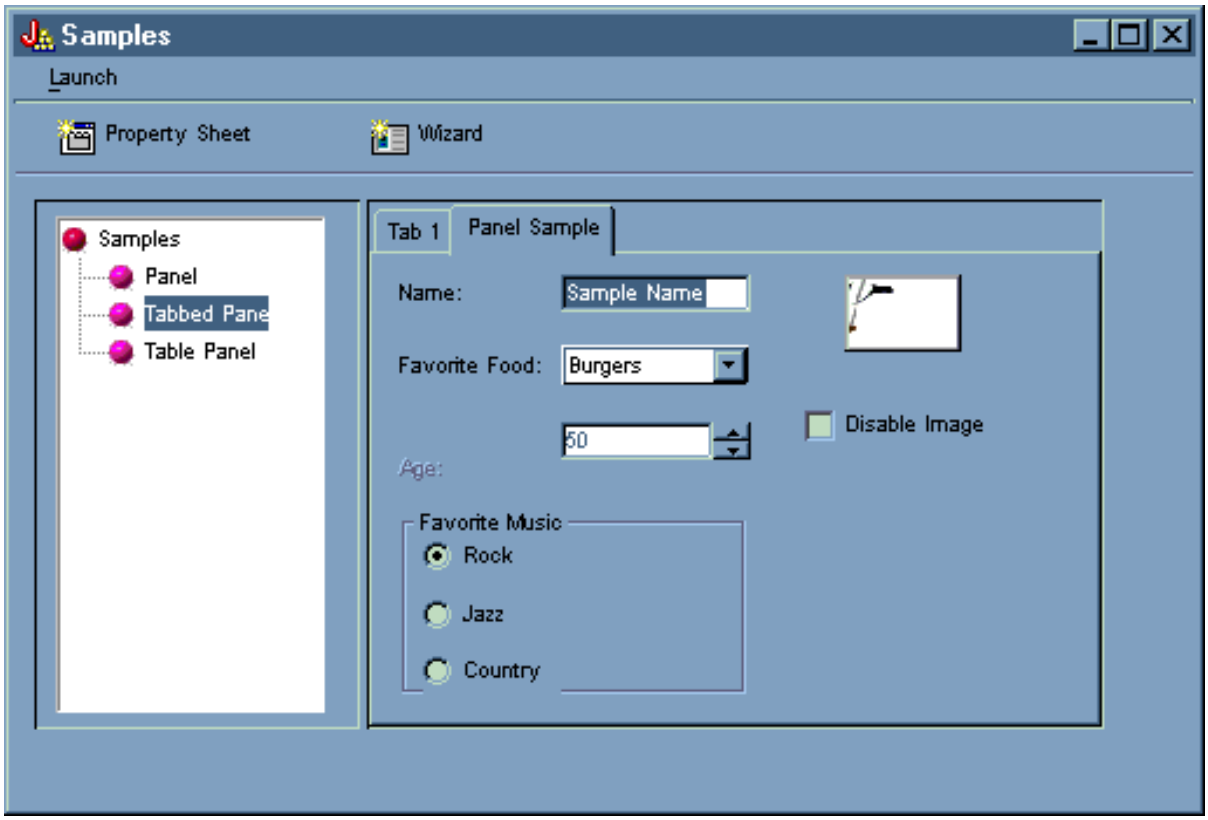
Select **Tab 1** again (in the right pane), then click **Disable Age Field on Tab 2** to deselect it.

Figure 20: Selecting Disable Age Field on Tab 2 in the right pane



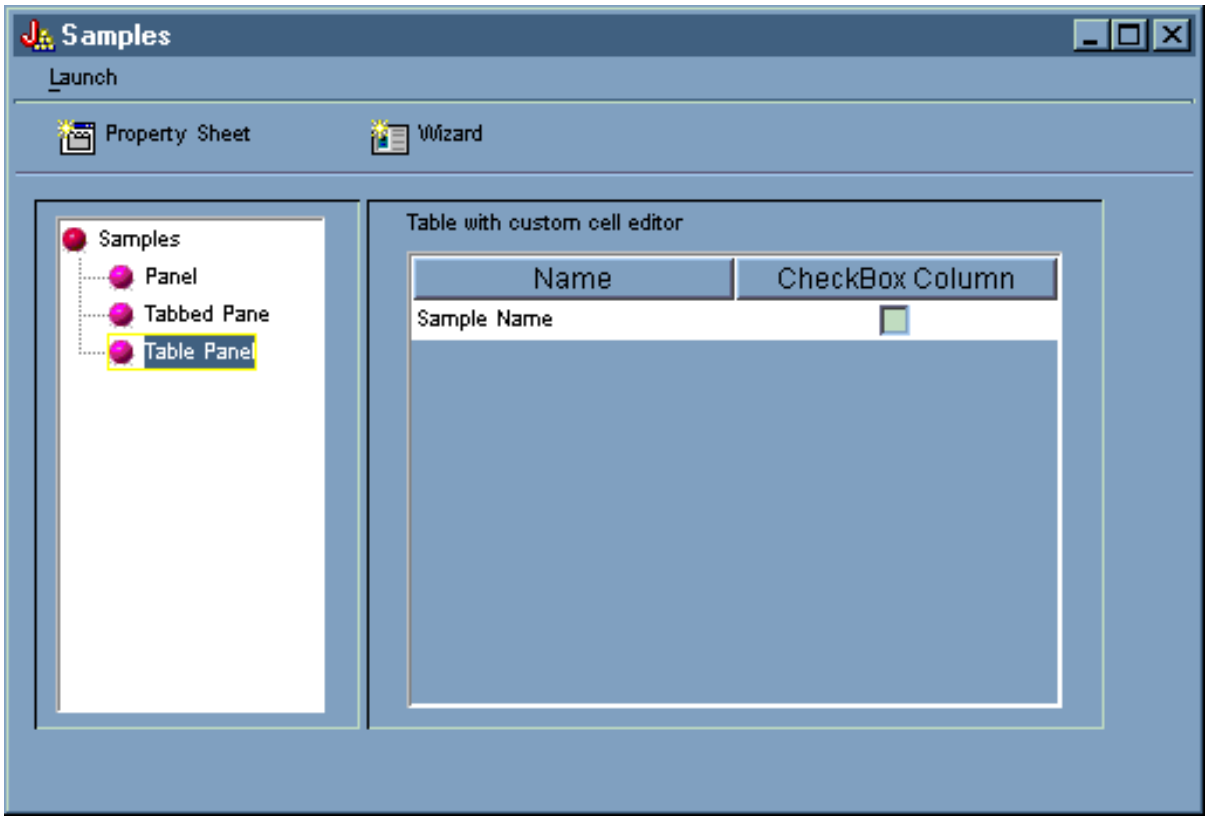
Selecting the **Disable Age Field on Tab 2** option deactivates and grays out the **Age** field in the **Panel Sample** tab, as shown in Figure 21.

Figure 21: Result of disabling age in the Panel Sample tab



Selecting **Table Pane** in the left pane of the GUI Builder example main window illustrates the use of a table panel with a custom renderer and a custom cell editor, as shown in Figure 22.

Figure 22: Selecting Table Panel in the left pane



Examples from the HTML classes

The following examples show you some of the ways that you can use the HTML classes:

- Example: Using the BidiOrdering class
- Example Creating HTMLAlign objects
- HTMLDocument class examples:
  - Example: Using HTMLDocument to create HTML data
  - Example: Using HTMLDocument to create XSL FO data
- Example: Using the HTML form classes
- Form input class examples:
  - Example: Creating a ButtonFormInput object
  - Example: Creating a FileFormInput object
  - Example: Creating a HiddenFormInput object
  - Example: Creating an ImageFormInput object
  - Example: Creating a ResetFormInput object
  - Example: Creating a SubmitFormInput object
  - Example: Creating a TextFormInput object
  - Example: Creating a PasswordFormInput object
  - Example: Creating a RadioFormInput object
  - Example: Creating a CheckboxFormInput object
- Example Creating HTMLHeading objects
- Example: Using the HTMLHyperlink class
- Example: Using the HTMLImage class
- HTMLList examples

  - Example: Creating ordered lists
  - Example: Creating unordered lists
  - Example: Creating nested lists
- Example: Creating HTMLMeta tags
- Example: Creating HTMLParameter tags
- Example: Creating HTMLServlet tags
- Example: Using the HTMLText class
- HTMLTree examples
  - Example: Using the HTMLTree class
  - Example: Creating a traversable integrated file system tree
- Layout form classes:
  - Example: Using the GridLayoutFormPanel class
  - Example: Using the LineLayoutFormPanel class
- Example: Using the TextAreaFormElement class
- Example: Using the LabelFormOutput class
- Example: Using the SelectFormElement class
- Example: Using the SelectOption class
- Example: Using the RadioFormInputGroup class
- Example: Using the RadioFormInput class
- Example: Using the HTMLTable classes
  - Example: Using the HTMLTableCell class
  - Example: Using the HTMLTableRow class
  - Example: Using the HTMLTableHeader class
  - Example: Using the HTMLTableCaption class

You can also use the HTML and servlet classes together, like in this example.

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Example: Using the HTML form classes

The following example shows you how to use the HTML form classes. You can also view a sample output from running this code. The HTML classes used in the ″showHTML″ method are **bold**.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML
```

```
// package classes, which allow you to easily build HTML Forms.
//
///////////////////////////////////////////////////////////////////////////

package customer;

import java.io.*;
import java.awt.Color;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.*;
import com.ibm.as400.util.html.*;


public class HTMLExample extends HttpServlet
{
  // 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 iSeries servers" from form
data = req.getParameter("list1");
if (data != null)
{
   output.write("Number of iSeries servers: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
//Getting "Comments" from form
data = req.getParameter("comments");
if (data != null && data.length() > 0)
{
   output.write("Comments: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
//Getting "Attachment"
data = req.getParameter("myAttachment");
if (data != null && data.length() > 0)
{
   output.write("Attachment File: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
//Getting Hidden "Copyright" infomation
data = req.getParameter("copyright");
if (data != null)
{
   output.write(data);
   output.newLine();
}
//-------------------------------------------------------------


output.flush();
output.close();

// Print a thanks to the customer
out.println("<HTML>");
out.println("<TITLE>Thank You!</TITLE>");
out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
out.println("<BODY BGCOLOR=\"blanchedalmond\">");
out.println("<HR><P>Thank You for Registering, <B>" + nameStr + "</B>!<P><HR>");
```

```
            // Create a HTMLHyperlink object and display it
            out.println("<UL><LI>" +
                        new HTMLHyperlink("./customer.HTMLExample", "Back to Registration Form"));
            out.println("</UL></BODY></HTML>");

          }

         }
        catch (Exception e)
        {
           // Show error in browser
           out.println("<HTML>");
           out.println("<TITLE>ERROR!</TITLE>");
           out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
           out.println("<BODY BGCOLOR=\"blanchedalmond\">");
           out.println("<BR><B>Error Message:</B><P>");
           out.println(e + "<P>");

           // Create a HTMLHyperlink object and display it
           out.println("<UL><LI>" +
                       new HTMLHyperlink("./customer.HTMLExample", "Back to Registration Form"));
           out.println("</UL></BODY></HTML>");

           e.printStackTrace();
        }
     }
     else
     {
        // Output a message to the customer saying customer name &
        // e-mail not entered. Please try again
        out.println ("<HTML> " +
                     "<TITLE>Invalid Registration Form</TITLE> " +
                     "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> " +
                     "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> " );

        out.println ("<HR><B>ERROR</B> in customer data - <P><B>" +
                     errorText +
                     "</B><P>Please Try Again... <HR>");


        // Create a HTMLHyperlink object and display it
        out.println("<UL><LI>" +
                    new HTMLHyperlink("./customer.HTMLExample", "Back to Registration Form") +
                    "</UL></BODY></HTML>");
     }
      // Close the writer
      out.close();


}


public void destroy(ServletConfig config)
{
   // do nothing
}


public String getServletInfo()
{
    return "My Product Registration";
}

private String showHTML()
{
  // String Buffer to hold HTML Page
```

```
StringBuffer page = new StringBuffer();

// Create the HTML Form object
HTMLForm form = new HTMLForm("/servlet/customer.HTMLExample");;
HTMLText txt;

// Build the beginning of the HTML Page and add it to the String Buffer
page.append("<HTML>\n");
page.append("<TITLE> Welcome!!</TITLE>\n");
page.append("<HEAD><SCRIPT LANGUAGE=\"JavaScript\">
            function test(){alert(\"This is a sample script executed with a
            ButtonFormInput.\")}</SCRIPT></HEAD>");
page.append("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\">\n");
page.append("<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"><BR>\n");

try
{
   //------------------------------------------------------------------
   // Create page title using HTML Text
   txt = new HTMLText("Product Registration");
   txt.setSize(5);
   txt.setBold(true);
   txt.setColor(new Color(199, 21, 133));
   txt.setAlignment(HTMLConstants.CENTER);

   // Add HTML Text to the String Buffer
   page.append(txt.getTag(true) + "<HR><BR>\n");
   //------------------------------------------------------------------

   //------------------------------------------------------------------
   // Create a Line Layout
   LineLayoutFormPanel line = new LineLayoutFormPanel();
   txt = new HTMLText("Enter your name and e-mail address:");
   txt.setSize(4);
   line.addElement(txt);

   // Add the Line Layout to String Buffer
   page.append(line.toString());
   page.append("<BR>");
   //------------------------------------------------------------------

   //------------------------------------------------------------------
   // Set the HTML Form METHOD
   form.setMethod(HTMLForm.METHOD_POST);
   //------------------------------------------------------------------

   //------------------------------------------------------------------
   // Create a Text input for the name.
   TextFormInput user = new TextFormInput("name");
   user.setSize(25);
   user.setMaxLength(40);

   // Create a Text input for the email address.
   TextFormInput email = new TextFormInput("email");
   email.setSize(30);
   email.setMaxLength(40);

   // Create a ImageFormInput
   ImageFormInput img =
     new ImageFormInput("Submit Form", "..\\images\\myPiimages/c.gif");
   img.setAlignment(HTMLConstants.RIGHT);
   //------------------------------------------------------------------

   //------------------------------------------------------------------
   // Create a LineLayoutFormPanel object for the name & e-mail address
   LineLayoutFormPanel line2 = new LineLayoutFormPanel();
```

```
// Add elements to the line form
line2.addElement(new LabelFormElement("Name:"));
line2.addElement(user);
// Create and add a Label Element to the Line Layout
line2.addElement(new LabelFormElement("E-mail:"));
line2.addElement(email);
line2.addElement(img);
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// Create Questions line layout
LineLayoutFormPanel line3 = new LineLayoutFormPanel();

// Add elements to the line layout
line3.addElement(new LineLayoutFormPanel());
line3.addElement(new
  CheckboxFormInput("use",
                    "yes",
                    "Do you currently use the Toolbox?",
                    false));
line3.addElement(new LineLayoutFormPanel());
line3.addElement(new CheckboxFormInput(
   "contact",
   "yes",
   "Would you like information on future Toolbox releases?",
   true));
line3.addElement(new LineLayoutFormPanel());
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// Create Version Radio Group
RadioFormInputGroup group = new RadioFormInputGroup("version");

// Add Radio Form Inputs to the Group
group.add(new RadioFormInput("version", "v3r2", "V3R2", false));
group.add(new RadioFormInput("version", "v4r1", "V4R1", false));
group.add(new RadioFormInput("version", "v4r2", "V4R2", false));
group.add(new RadioFormInput("version", "v4r3", "V4R3", false));
group.add(new RadioFormInput("version", "v4r4", "V4R4", false));
group.add(new
  RadioFormInput("version",
                 "multiple versions",
                 "Multiple Versions? Which ones:",
                 false));

//Create a Select Form Element
SelectFormElement mlist = new SelectFormElement("MultiList");
mlist.setMultiple(true);
mlist.setSize(3);

//Create the Options for the Select Form Element
SelectOption option1 = mlist.addOption("V3R2", "v3r2");
SelectOption option2 = mlist.addOption("V4R1", "v4r1");
SelectOption option3 = mlist.addOption("V4R2", "v4r2");
SelectOption option4 = mlist.addOption("V4R3", "v4r3");
SelectOption option5 = mlist.addOption("V4R4", "v4r4");

// Create HTML text
txt = new HTMLText("Current Server Level:");
txt.setSize(4);

// Create Grid Layout
GridLayoutFormPanel grid1 = new GridLayoutFormPanel(3);

// Add radio group & select form element to the grid
grid1.addElement(txt);
grid1.addElement(group);
```

```
grid1.addElement(mlist);
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Create Grid Layout for interests
GridLayoutFormPanel grid2 = new GridLayoutFormPanel(1);
txt = new HTMLText("Current Projects or Area of Interest: " +
                   "(check all that apply)");
txt.setSize(4);

// Add elements to Grid Layout
grid2.addElement(new LineLayoutFormPanel());
grid2.addElement(txt);
// Create and add a Checkbox to the Grid Layout
grid2.addElement(new
  CheckboxFormInput("interest", "applications", "Applications", true));
grid2.addElement(new
  CheckboxFormInput("interest", "applets", "Applets", false));
grid2.addElement(new
  CheckboxFormInput("interest", "servlets", "Servlets", false));
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Create Line Layout for platforms
LineLayoutFormPanel line4 = new LineLayoutFormPanel();
txt = new HTMLText("Client Platforms Used: " +
                   "(check all that apply)");
txt.setSize(4);

// Add elements to Line Layout
line4.addElement(new LineLayoutFormPanel());
line4.addElement(txt);
line4.addElement(new LineLayoutFormPanel());
line4.addElement(new CheckboxFormInput("platform",
                                       "95",
                                       "Windows95",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "98",
                                       "Windows98",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "NT",
                                       "WindowsNT",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "OS2",
                                       "OS/2",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "AIX",
                                       "AIX",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "Linux",
                                       "Linux",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "AS400",
                                       "iSeries",
                                       false));
line4.addElement(new CheckboxFormInput("platform",
                                       "Other",
                                       "Other:",
                                       false));

TextFormInput other = new TextFormInput("OtherPlatforms");
```

```
other.setSize(20);
other.setMaxLength(50);

line4.addElement(other);
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// Create a Line Layout for number of servers
LineLayoutFormPanel grid3 = new LineLayoutFormPanel();

txt = new HTMLText(
   "How many iSeries 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 class example output

These are some possible sample outputs you may get from running the HTML class example:

- 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 iSeries servers: three
Comments: The Toolbox is being used by our entire Programming department to build customer applications!
Attachment File: U:\wiedrich\servlet\temp.html

- 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 iSeries servers: FiveOrMore

- 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 iSeries servers: one
  Comments: This is my first time using this! Very Cool!

- Customer Name: Clark Kent
  Email address: superman@krypton.com
  AS400 Version: v4r2
  Number of iSeries servers: one

Example: Using HTMLTree classes

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// 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";
   }
}
```

Example: Creating a traversable integrated file system tree (File one of three)

This example code, in conjunction with the code in the other two example files, displays an HTMLTree and FileListElement in a servlet. The three files in the example are:

- FileTreeExample.java - this file, which generates the HTML frames and starts the servlet
- TreeNav.java - builds and manages the tree
- TreeList.java - displays the contents of selections made in the TreeNav.java class

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML package
// classes, which allow you to easily build HTML and File Trees.
//
///////////////////////////////////////////////////////////////////////////

import java.io.PrintWriter;
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.HTMLMeta;


//
// An example of using frames to display an HTMLTree and FileListElement
// in a servlet.
//
```

```
public class FileTreeExample extends HttpServlet
{
   public void init(ServletConfig config)
      throws ServletException
   {
      super.init(config);
   }

   /**
    *  Process the GET request.
    *  @param req The request.
    *  @param res The response.
    **/

   public void doGet (HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
   {
      resp.setContentType("text/html");

      // Set up two frames. The first, a navigation frame, will display
      // the HTMLTree, which will contain FileTreeElements and allow
      // navigation of the File system.  The second frame will display/list
      // the contents of a selected directory from the navigation frame.
      PrintWriter out = resp.getWriter();
      out.println("<html>\n");
      out.println(new HTMLMeta("Expires","Mon, 04 Jan 1990 13:00:00 GMT"));
      out.println("<frameset cols=\"25%,*\">");
      out.println("<frame frameborder=\"5\" src=\"/servlet/TreeNav\" name=\"nav\">");
      out.println("<frame frameborder=\"3\" src=\"/servlet/TreeList\" name=\"list\">");
      out.println("</frameset>");
      out.println("</html>\n");
      out.close();
   }

   /**
    *  Process the POST request.
    *  @param req The request.
    *  @param res The response.
    **/

   public void doPost (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
      res.setContentType("text/html");
      ServletOutputStream out = res.getOutputStream();
   }

   public void destroy(ServletConfig config)
   {
      // do nothing
   }

   public String getServletInfo()
   {
      return "FileTree Servlet";
   }
}
```

Example: Creating a traversable integrated file system tree (File two of three)

This example code, in conjunction with the code in the other two example files, displays an HTMLTree and FileListElement in a servlet. The three files in the example are:

- FileTreeExample.java - generates the HTML frames and starts the servlet
- TreeNav.java - this file, which builds and manages the tree

- TreeList.java - displays the contents of selections made in the TreeNav.java class

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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";
    }
}
```

Example: Creating a traversable integrated file system tree(File three of three)

This example code, in conjunction with the code in the other two example files, displays an HTMLTree and FileListElement in a servlet. The three files in the example are:

- FileTreeExample.java - generates the HTML frames and starts the servlet
- TreeNav.java - builds and manages the tree
- TreeList.java - this file, which displays the contents of selections made in the TreeNav.java class

Note: Read the Code example disclaimer for important legal information.

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

Example: Using HTMLTable classes

The following example shows you how the HTMLTable classes work:

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

The Java code example above generates the following HTML code:

```
<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>
```

The following table shows how the HTML code above displays in a Web browser.

Table 2. Customer Account Balances - January 1, 2000

ACCOUNT	NAME	BALANCE
0000001	Customer1	100.00
0000002	Customer2	200.00
0000003	Customer3	550.00

Examples: Program Call Markup Language (PCML)

The following examples use Program Call Markup Language to call i5/OS APIs, and each one links to a page that shows the PCML source followed by a Java program.

- Simple example of retrieving data: Shows the PCML source and Java program needed to retrieve information about a user profile on the server. The API being called is the *Retrieve User Information* (**QSYRUSRI**) API.
- Retrieving a list of information: Shows the PCML source and Java program needed to retrieve a list of authorized users on a server. The API being called is the *Open List of Authorized Users* (**QGYOLAUS**) API. This example illustrates how to access an array of structures returned by a server program.
- Retrieving multidimensional data: Shows the PCML source and Java program needed to retrieve a list Network File System (NFS) exports from a server. The API being called is the *Retrieve NFS Exports* (**QZNFRTVE**) API. This example illustrates how to access arrays of structures within an array of structures.

Note: The proper authority for each example varies but may include specific object authorities and special authorities. In order to run these examples, you must sign on with a user profile that has authority to perform the following actions:

- Call the i5/OS API in the example
- Access the information being requested

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Simple example of retrieving data

Note: Read the Code example disclaimer for important legal information.

This simple example has two parts:

- PCML source for calling QSYRUSRI
- Java program source for calling QSYRUSRI

PCML source for calling QSYRUSRI

```
<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>
```

Java program source for calling QSYRUSRI

```
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()

}
```

Example: Retrieving a list of information

This example has two parts:

- PCML source for calling QGYOLAUS
- Java program source for calling QGYOLAUS

PCML source for calling QGYOLAUS

```
<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>
```

Java program source for calling QGYOLAUS

```
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);
 }
}
```

Example: Retrieving multidimensional data

This example has two parts:

- PCML source for calling QZNFRTVE
- Java program source for calling QZNFRTVE

PCML source for calling QZNFRTVE

```
<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>
```

Java program source for calling QZNFRTVE

```
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()
}
```

Examples: ReportWriter classes

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java reportwriter classes.

JSPReportProcessor and PDFContext

- Example: Using JSPReportProcessor with PDFContext
- Example: JSPReportProcessor sample JSP file

XSLReportProcessor and PCLContext

- Example: Using XSLReportProcessor with PCLContext
- Example: XSLReportProcessor sample XML file
- Example: XSLReportProcessor sample XSL file

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Using JSPReportProcessor with PDFContext

Note: Read the Code example disclaimer for important legal information.

To view the contents of an example JSP source file that you can use with JSPRunReport, see JSPcust_table.jsp. You can also download a ZIP file that contains the JSP example file. The zipped file also contains XML and XSL example files that you can use with the XSLReportProcessor example (PCLRunReport).

```
////////////////////////////////////////////////////////////////////////////
//
// The following example (JSPRunReport) uses the JSPReportProcessor and the
// PDFContext classes to obtain data from a specified URL and convert the data
// to the PDF format. The data is then streamed to a file as a PDF document.
//
// 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);
    }
}
```

Example: JSPReportProcessor sample JSP file

Note: Read the Code example disclaimer for important legal information.

```
<?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>
```

Example: Using XSLReportProcessor with PCLContext

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////////
//
// The following example (PCLRunReport) uses the XSLPReportProcessor and the
// PCLContext classes to obtain XML data and convert the data to the PCL format.
// The data is then streamed to a printer OutputQueue.
//
// To view the contents of example XML and XSL source files that you can use
// with PCLRunReport, see realestate.xml and realestate.xsl. You can also
// download a zip file that contains the XML and XSL example files. The zip
// file also contains a JSP example file that you can use with the
// JSPReportProcessor example (JSPRunReport).
//
// Command syntax:
//     java PCLRunReport <xml_file> <xsl_file>
//
////////////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.awt.*;
import java.io.*;
import java.awt.print.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.HashMap;

import com.ibm.xsl.composer.flo.*;
import com.ibm.xsl.composer.areas.*;
import com.ibm.xsl.composer.framework.*;
import com.ibm.xsl.composer.java2d.*;
import com.ibm.xsl.composer.prim.*;
import com.ibm.xsl.composer.properties.*;
import com.ibm.as400.util.reportwriter.processor.*;
import com.ibm.as400.util.reportwriter.pclwriter.*;
import java.io.IOException;
import java.io.Serializable;
import org.xml.sax.SAXException;
import com.ibm.as400.access.*;

public class PCLRunReport

{

    public static void main(String args[])
    {
        SpooledFileOutputStream fileout = null;
        String xmldocumentName = args[0];
        String xsldocumentName = args[1];

        String sys = "<system>";      /* Insert ISeries server name       */
        String user = "<user>";       /* Insert ISeries user profile name */
        String pass = "<password>";   /* Insert ISeries password          */

        AS400 system = new AS400(sys, user, pass);

        /* Insert ISeries output queue */
        String outqname = "/QSYS.LIB/qusrsys.LIB/<outq>.OUTQ";
```

```
OutputQueue outq = new OutputQueue(system, outqname);
PrintParameterList parms = new PrintParameterList();
parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outq.getPath());

try{
  fileout = new SpooledFileOutputStream(system, parms, null, null);
}
catch (Exception e)
{}

/** set up page format **/
Paper paper = new Paper();
paper.setSize(612,792);
paper.setImageableArea(18, 36, 576, 720);
PageFormat pf = new PageFormat();
pf.setPaper(paper);

/** create a PCLContext object and case FileOutputStream
    as an OutputStream **/
PCLContext pclcontext = new PCLContext((OutputStream)fileout, pf);

System.out.println("Ready to parse XSL document");

/** create the XSLReportProcessor object **/
XSLReportProcessor xslprocessor = new XSLReportProcessor(pclcontext);
try {
xslprocessor.setXMLDataSource(xmldocumentName);
}
catch (SAXException se) {
    String mes = se.getMessage();
    System.out.println(mes);
    System.exit(0);
    }
catch (IOException ioe) {
    String mes = ioe.getMessage();
    System.out.println(mes);
    System.exit(0);
    }
catch (NullPointerException np){
    String mes = np.getMessage();
    System.out.println(mes);
    System.exit(0);
    }
/** set the template to the specified XML data source **/
try {
xslprocessor.setTemplate(xsldocumentName);
}
catch (NullPointerException np){
    String mes = np.getMessage();
    System.out.println(mes);
    System.exit(0);
    }
catch (IOException e) {
    String mes = e.getMessage();
    System.out.println(mes);
    System.exit(0);
    }
catch (SAXException se) {
    String mes = se.getMessage();
    System.out.println(mes);
    System.exit(0);
    }

/** process the report **/
try {
xslprocessor.processReport();
}
```

```
        catch (IOException e) {
            String mes = e.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        catch (SAXException se) {
            String mes = se.getMessage();
            System.out.println(mes);
            System.exit(0);
            }

        System.exit(0);
    }

}
```

Example: XSLReportProcessor sample XML file

Note: Read the Code example disclaimer for important legal information.

```
<?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>
```

Example: XSLReportProcessor sample XSL file

Note: Read the Code example disclaimer for important legal information.

```
<?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>
```

Examples: Resource classes

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java resource classes.

Resource and ChangeableResource

- Example: Retrieving an attribute value from RUser, a concrete subclass of Resource
- Example: Setting attribute values for RJob, a concrete subclass of ChangeableResource
- Example: Using generic code to access resources

ResourceList

- Example: Getting and printing the contents of a ResourceList
- Example: Using generic code to access a ResourceList
- Example: Presenting a resource list in a servlet

Presentation

- Example: Using Presentations

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Examples: Resource list

The following examples show various ways of working with resource lists:

- Example: Getting and printing the contents of a ResourceList
- Example: Using generic code to access a ResourceList
- Example: Presenting a resource list in a servlet

Example: Getting and printing the contents of a ResourceList

One example of a concrete subclass of ResourceList is com.ibm.as400.resource.RJobList, which represents a list of iSeries jobs. RJobList supports many selection IDs and sort IDs, each of which can be used to filter or sort the list.This example prints the contents of an RJobList:

```
// 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();
```

Code example disclaimer

The following disclaimer applies to all of the IBM Toolbox for Java examples:

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Presenting a resource list in a servlet

Use the ResourceListRowData class in conjunction with the HTMLFormConverter or HTMLTableConverter class to present a resource list in a servlet.

- HTMLFormConverter displays the contents of the resource list as a series of forms, where each form contains attribute values for a resource in the resource list.
- HTMLTableConverter displays the contents of the resource list as a table, with each row contains information about a resource in the resource list.

The columns for a ResourceListRowData object are specified as an array of column attribute IDs while each row represents a resource object.

```
// 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]);
```

Example: Retrieving an attribute value from a Resource

One concrete subclass of Resource is com.ibm.as400.resource.RUser, which represents an iSeries user. RUser supports many attribute IDs, each of which you can use to get attribute values.

This example retrieves an attribute value from an RUser:

```
// 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);
```

Example: Setting attribute values for a ChangeableResource

One concrete subclass of ChangeableResource is com.ibm.as400.resource.RJob, which represents an iSeries job. RJob supports many attribute IDs, each of which you can use to access attribute values. This example sets two attribute values for an RJob:

```
// 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();
```

Example: Using generic code to access resources

You can write generic code to work with any Resource, ResourceList or ChangeableResource subclass. Such code may improve reusability and maintainability and will work with future Resource, ResourceList or ChangeableResource subclasses without modification.

Every attribute has an associated attribute meta data object (com.ibm.as400.resource.ResourceMetaData) that describes various properties of the attribute. These properties include whether or not the attribute is read only and what the default and possible values are.

Examples:

Example: Printing the contents of a ResourceList

Here is an example of generic code that prints some of the contents of a ResourceList:

```
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));
    }
}
```

Example: Using ResourceMetaData to access every attribute supported by a resource

This is an example of generic code that prints the value of every attribute supported by a resource:

```
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();
}
```

Examples: RFML

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java RFML component:

- Example: Using RFML compared to using IBM Toolbox for Java Record classes
- Example: RFML source file

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: RFML source file

This example RFML source file defines the format of customer records as used in the RFML example Using RFML compared to using IBM Toolbox for Java Record classes. This RFML source file might be a text file named qcustcdt.rfml.

Note: Read the Code example disclaimer for important legal information.

```
<?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>
```

Example: Using a profile token credential to swap the i5/OS thread identity

Note: Read the Code example disclaimer for important legal information.

The following code example shows you how to use a profile token credential to swap the i5/OS thread identity and perform work on behalf of a specific user:

```
// Prepare to work with the local AS/400 system.
AS400 system = new AS400("localhost", "*CURRENT", "*CURRENT");

// Create a single-use ProfileTokenCredential with a 60 second timeout.
// A valid user ID and password must be substituted.
ProfileTokenCredential pt = new ProfileTokenCredential();
pt.setSystem(system);
```

```
pt.setTimeoutInterval(60);
pt.setTokenType(ProfileTokenCredential.TYPE_SINGLE_USE);

pt.setTokenExtended("USERID", "PASSWORD");

// Swap the i5/OS thread identity, retrieving a credential to
// swap back to the original identity later.
AS400Credential cr = pt.swap(true);

// Perform work under the swapped identity at this point.

// Swap back to the original i5/OS thread identity.
cr.swap();

// Clean up the credentials.
cr.destroy();
pt.destroy();
```

Examples from the servlet classes

The following examples show you some of the ways that you can use the servlet classes:

- Example: Using the ListRowData class
- Example: Using the RecordListRowData class
- Example: Using the SQLResultSetRowData class
- Example: Using the HTMLFormConverter class
- Example: Using the ListMetaData class
- Example: Using the SQLResultSetMetaData class
- Example: Presenting a resource list in a servlet

You can also use the servlet and HTML classes together, like in this example.

Code example disclaimer

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
>
> All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Using ListRowData

This example has three parts:

- “Java source that shows how the ListRowData class works” on page 620
- “HTML source generated from the Java source by the using HTMLTableConverter” on page 620
- “How a browser displays the generated HTML” on page 621

Java source that shows how the ListRowData class works

```
// 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]);
```

HTML source generated from the Java source by the using HTMLTableConverter

Using the "HTMLTableConverter class" on page 229 in the Java source example above generates the following HTML code.

```
<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>
```

How a browser displays the generated HTML

The following table shows how the HTML source code looks when viewed in a browser.

Customer ID	Order Number
777-53-4444	12345-XYZ
777-53-4444	56789-ABC

Example: Using RecordListRowData

This example has three parts:

- Java source that shows how the RecordListRowData class works
- HTML source generated from the Java source by the using HTMLTableConverter
- How a browser displays the generated HTML

Java source that shows how the RecordListRowData class works

```
// 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]);
```

HTML source generated from the Java source by the using HTMLTableConverter

Using the HTMLTableConverter class in the Java source example above generates the following HTML code.

```
<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>
```

How a browser displays the generated HTML

The following table shows how the HTML source code looks when viewed in a browser.

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

Example: Using SQLResultSetRowData

This example has three parts:

- Java source that shows how the SQLResultSetRowData class works
- HTML source generated from the Java source by the using HTMLTableConverter

- How a browser displays the generated HTML

Java source that shows how the SQLResultSetRowData class works

```
// 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]);
```

HTML source generated from the Java source by the using HTMLTableConverter

Using the HTMLTableConverter class in the Java source example above generates the following HTML code.

```
<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>
```

How a browser displays the generated HTML

The following table shows how the HTML source code looks when viewed in a browser.

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

Example: Using HTMLFormConverter

While running a web server with servlet support, compile and run the following example to see how the HTMLFormConverter works:

```
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();
   }

}
```

The HTML generated by the above example looks like this:

```
<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>
```

Lights On example for the HTML and servlet classes

This example shows you how the HTML and servlet classes work. It is a general overview. To view this example, compile and run it with a webserver and browser running.

```
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();
  }

}
```

Simple programming examples

These examples show some of the ways you can begin to code your own Java programs using the IBM Toolbox for Java classes. Meant for programmers just beginning to use IBM Toolbox for Java classes, these examples include detailed explanations about key lines in the code.

If you want some help getting started, see Writing your first IBM Toolbox for Java program.

For links to many of the other examples provided in the IBM Toolbox for Java information, see Code examples.

Use the following list to view the simple programming examples:

- Calling commands
- Using message queues
- Using record-level access
- Using JDBC classes to create and populate a table
- Displaying a list of server jobs in a GUI

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Writing your first IBM Toolbox for Java program

To begin this simple exercise, you must have installed Java on your workstation. You can decide which version you want to install by reviewing Requirements for running Java applications.

After you have Java installed on your client, complete the following tasks:

1. Copy jt400.jar to the workstation.
2. Add jt400.jar to the CLASSPATH on your workstation by appending the full path of the JAR file to the CLASSPATH. For example, when the jt400.jar file resides in the c:\lib directory on your workstation (running Windows), add the following to the end of your CLASSPATH statement:

   ```
   ;c:\lib\jt400.jar
   ```
3. Open a text editor and type the first simple programming example

   Note: Make sure to leave out the text that refers to the Notes (for example, Note 1, Note 2, and so on). Save the new document with the name CmdCall.java.
4. Start a command session on your workstation and use the following command to compile the simple programming example:

   ```
   javac CmdCall.java
   ```
5. In the command session, type the following command to run the simple programming example:

   ```
   java CmdCall
   ```

[Simple programming examples]

Example: Using CommandCall

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// Example using the IBM Toolbox for Java's access class, CommandCall.
//
// This source is an example of IBM Toolbox for Java "Job List".
//
////////////////////////////////////////////////////////////////////////////
//
// The access classes of IBM Toolbox for Java 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 uses the ″AS400″ object to identify the target server. If you construct the AS400 object with no parameters, IBM Toolbox for Java prompts for the system name, userid and password. The AS400 class also includes a constructor that takes the system name, userid and password.
2. Use the IBM Toolbox for Java CommandCall object to send commands to the server. When you create the CommandCall object, you pass it an AS400 object so that it knows which server is the target of the command.
3. Use the run() method on the command call object to run a command.
4. The result of running a command is a list of i5/OS messages. IBM Toolbox for Java represents these messages as AS400Message objects. When the command is complete, you get the resulting messages from the CommandCall object.
5. Print the message text. Also available are the message ID, message severity, and other information. This program prints only the message text.

Example: Using message queues (part 1 of 3)

[Next part]

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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. This class is in the 'examples' package. Java uses packages to avoid name conflicts between Java class files.
2. This line makes all of the IBM Toolbox for Java classes in the access package available to this program. The classes in the access package have the common prefix **com.ibm.as400**. By using an import statement, the program can refer to a class using just its name, not its fully-qualified name. For example, you can reference the AS400 class by using AS400, instead of com.ibm.as400.AS400.
3. This class has a **main** method; therefore, it can be run as an application. To invoke the program, you run **java examples.displayMessages**. Note that case must match when running the program. Because an IBM Toolbox for Java class is used, jt400.zip must be in the classpath environment variable.
4. The main method mentioned in Note 3 is static. One of the restrictions of static methods is that static methods can call only other static methods in their class. To avoid this restriction, many java programs create an object, and then do initialization processing in a method called **Main**. The Main() method can call any other method in the displayMessages object.
5. The IBM Toolbox for Java creates threads on behalf of the application to carry out IBM Toolbox for Java activity. If the program does not issue **System.exit(0)** at termination time, the program may not terminate normally. For example, suppose this program was run from a Windows 95 DOS prompt. Without this line, the command prompt does not return when the program finished. The user has to enter Ctrl-C to get a command prompt.
6. The IBM Toolbox for Java code throws exceptions that your program must catch.
7. This program displays the text of the exception while the program is performing error processing. Exceptions thrown by the IBM Toolbox for Java are translated, so the text of the exception will be the same as the language of the workstation.

[Next part]

Example: Using message queues (part 2 of 3)

[Previous part | Next part]

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// 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. A program uses the **AS400** object to designate which server to connect to. With one exception, all programs that need resources from a server must have an AS400 object. The exception is JDBC. If your program uses JDBC, then the IBM Toolbox for Java JDBC driver creates the AS400 object for the program.

2. This program assumes the first command line parameter is the name of the server. If a parameter is passed to the program, the **setSystemName** method of the AS400 object is used to set the system name. The AS400 object also needs server sign-on information:
   - If the program is running on a workstation, the IBM Toolbox for Java program prompts the user for a user ID and password. **Note:** If a system name is not specified as a command line parameter, the AS400 object also prompts for the system name.
   - If the program is running on the iSeries JVM, then the user ID and password of the user running the Java program is used. In this case, the user does not specify a system name, but lets the system name default to the name of the system that the program is running on.

[Previous part | Next part]

Example: Using message queues (part 3 of 3)

[Previous part]

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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. The purpose of this program is to display messages in a server message queue. The **MessageQueue** object of the IBM Toolbox for Java is used for this task. When the message queue object is constructed, the parameters are the AS400 object and the message queue name. The AS400 object indicates which server contains the resource, and the message queue name identifies which message queue on the server. In this case, a constant is used, which tells the message queue object to access the queue of the signed-on user.
2. The message queue object gets a list of messages from the server. A connection to the server is made at this point.
3. Remove a message from the list. The message is in the IBM Toolbox for Java program's QueuedMessage object.
4. Print the text of the message.

[Previous part]

Example: Using record-level access (part 1 of 2)

[Next part]

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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. This line of code creates an AS400 object and connects to the record-level access service.
2. This line creates a QSYSObjectPathName object that obtains the integrated file system path name form of the object to be displayed.
3. This statement creates an object that represents an existing sequential file on the server you are connected to. This sequential file is the file that will be displayed.
4. These lines retrieve the record format of the file.
5. This line sets the record format for the file.
6. This line opens the selected file for reading. It will read 100 records at a time, when it is possible.
7. This line of code reads each record in sequence.
8. This line closes the file.
9. This line disconnects from the record-level access service.

[Next part]

Example: Using record-level access (part 2 of 2)

[Previous part]

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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. (args[0]) in the previous line and MYFILE.FILE are pieces of code that are prerequisites for the rest of the example to run. The program assumes that the library MYLIB exists on the server and that the user has access to it.
2. The text within the Java comments labeled ″Begin Note Two″ and ″End Note Two″ shows how to create a record format yourself instead of getting the record format from an existing file. The last line in this block creates the file on the server.
3. The text within the Java comments labeled ″Begin Note Three″ and ″End Note Three″ shows a way to create a record and then write it to a file.

[Previous part]

Example: Using JDBC classes to create and populate a table (part 1 of 2)

[Next part]

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
//////////////////////////////////////////////////////////////////////////////
//
// 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. This line loads the IBM Toolbox for Java JDBC driver. A JDBC driver is necessary to mediate between JDBC and the database you are working with.
2. This statement connects to the database. A prompt will appear for the user ID and password. A default schema is provided so that you will not need to qualify the table name in SQL statements.
3. These lines delete the table if it already exists.
4. These lines create the table.
5. This line prepares a statement that will insert rows into the table. Because you will be executing this statement several times, you should use a PreparedStatement and parameter markers.
6. This block of code populates the table for you; every time the loop is executed, it inserts a row into the table.
7. Now that the table has been created and filled in, this statement closes the connection to the database.

[Next part]

Example: Using JDBC classes to create and populate a table (part 2 of 2)

[Previous part]

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// 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. This line loads the IBM Toolbox for Java JDBC driver. A JDBC driver mediates between JDBC and the database with which you are working.
2. This line retrieves the connection's meta data, an object that describes many of the characteristics of the database.
3. This statement executes the query on the specified table.
4. These lines retrieve information about the table.
5. These lines set the column width to either the length of the label or the length of the data, whichever is longer.
6. This block of code iterates through all of the rows in the table and displays the contents of each column in each row.

[Previous part]

Example: Displaying a list of server jobs in a GUI

Use the following as an example for your program.

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// Example using the IBM Toolbox for Java's vaccess
// class, VJobList.
//
// This source is an example of IBM Toolbox for Java "Job List".
//
```

```
////////////////////////////////////////////////////////////////////////////////

package examples;  Note 1

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;  Note 2

import javax.swing.*;  Note 3
import java.awt.*;
import java.awt.event.*;

public class GUIExample
{

  public static void main(String[] parameters)  Note 4
  {
     GUIExample example = new GUIExample(parameters);

  }

  public GUIExample(String[] parameters)
  {
     try  Note 5
     {
          // Create an AS400 object.
     //The system name was passed as the first command line argument.
          AS400 system = new AS400 (parameters[0]);  Note 6

          VJobList jobList = new VJobList (system);  Note 7

     // Create a frame.
     JFrame frame = new JFrame ("Job List Example");  Note 8

     // Create an error dialog adapter.  This will display any errors to the user.
     ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (frame);  Note 9

     // Create an explorer pane to present the job list.
     AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList);  Note 10

     explorerPane.addErrorListener (errorHandler);  Note 11

     // Use load to load the information from the system.
     explorerPane.load();  Note 12

     // When the frame closes, exit the program.
     frame.addWindowListener (new WindowAdapter ()  Note 13
     {
           public void windowClosing (WindowEvent event)
           {
              System.exit(0);
           }
     } );

     // Layout the frame with the explorer pane.
     frame.getContentPane().setLayout(new BorderLayout() );
     frame.getContentPane().add("Center", explorerPane);  Note 14

     frame.pack();
     frame.show();  Note 15
     }

     catch (Exception e)
     {
          e.printStackTrace();  Note 16
     }
```

```
    System.exit(0);  Note 17

    }
}
```

1. This class is in the examples package. Java uses packages to avoid name conflicts between Java class files.
2. This line makes all of the IBM Toolbox for Java classes in the vaccess package available to this program. The classes in the vaccess package have the common prefix com.ibm.as400.vaccess. By using an import statement, the program calls the name instead of the package plus name. For example, you can reference the AS400ExplorerPane class by using AS400ExplorerPane, not com.ibm.as400.AS400ExplorerPane.
3. This line makes all of the Java Foundation Classes (JFC) in the Swing package available to this program. Java programs that use the IBM Toolbox for Java vaccess (GUI) classes need JDK 1.1.2 plus Java Swing 1.0.3 from Sun Microsystems, Inc. Swing is available with Sun's JFC 1.1.
4. This class has a main method so it can be run as an application. To invoke the program, run "java examples.GUIExample serverName", where serverName is the name of your server. Either the jt400.zip or jt400.jar must be in your classpath for this to work.
5. The IBM Toolbox for Java code throws exceptions that your program must catch.
6. The AS400 class is used by IBM Toolbox for Java. This class manages sign-on information, creates and maintains socket connections, and sends and receives data. In this example, the program will pass the server name to the AS400 object.
7. The VJobList class is used by the IBM Toolbox for Java to represent a list of server jobs that can be displayed in a vaccess (GUI) component. Notice that the AS400 object is used to specify the server on which the list resides.
8. This line constructs a frame or a top-level window that will be used to display the job list.
9. ErrorDialogAdapter is an IBM Toolbox for Java graphical user interface (GUI) component that is created to automatically display a dialog window whenever an error event occurs in the application.
10. This line creates an AS400ExplorerPane, a graphical user interface (GUI) that represents a hierarchy of objects within a server resource. The AS400ExplorerPane presents a tree on the left side rooted at the VJobList and the details of the resource in the right side. This only initializes the pane to a default state and does not load the contents of the VJobList to the pane.
11. This line adds the error handler you created in step nine as a listener on the VJobList graphical user interface (GUI) component.
12. This line loads the contents of the JobList into the ExplorerPane. This method must be called explicitly to communicate to and load information from the server. This gives the application control over when the communication with the server will occur. With this you can:
    - Load the contents before adding the pane to a frame. The frame does not appear until all the information is loaded, as in this example.
    - Load the contents after adding the pane to a frame and displaying that frame. The frame appears with a "wait cursor" and the information is filled in as it is loaded.
13. This line adds a window listener so that the application ends when the frame closes.
14. This line adds the job list graphical user interface GUI component to the center of the controlling frame.
15. This line calls the show method to make the window visible to the user.
16. IBM Toolbox for Java exceptions are translated so the text will appear in the language of the workstation. For example, this program displays the text of the exception as its error processing.
17. The IBM Toolbox for Java creates threads to carry out IBM Toolbox for Java activity. If the program does not do System.exit(0) when it is terminated, the program may not exit normally. For example, if the program was run from a Windows 95 DOS prompt without this line, the command prompt does not return when the program finished.

Examples: Tips for programming

This section lists the code examples that are provided throughout the documentation of IBM Toolbox for Java tips for programming.

Managing connections

- Example: Making a connection to the iSeries server with a CommandCall object
- Example: Making two connections to the iSeries server with a CommandCall object
- Example: Creating CommandCall and IFSFileInputStream objects with an AS400 object
- Example: Using AS400ConnectionPool to preconnect to the iSeries server
- Example: Using AS400ConnectionPool to preconnect to a specific service on the iSeries server, then reuse the connection

Starting and ending connections

- Example: How a Java program preconnects to an iSeries server
- Example: How a Java program disconnects from an iSeries server
- Example: How a Java program disconnects and reconnects to the iSeries server with disconnectService() and run()
- Example: How a Java program disconnects from the iSeries server and fails to reconnect

Exceptions

- Example: Using exceptions

Error events

- Example: Handling error events
- Example: Defining an error listener
- Example: Using a customized handler to handle error events

Trace

- Example: Using trace
- Example: Using setTraceOn()
- Example: Using component trace

Optimization

- Example: Creating two AS400 objects
- Example: Using an AS400 object to represent a second server

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Examples: ToolboxME for iSeries

This section lists the code examples that are provided throughout the IBM ToolboxME for iSeries documentation.

- "ToolboxME for iSeries example: JdbcDemo.java" on page 348
- "Example: Using ToolboxME for iSeries, MIDP, and JDBC"
- "Example: Using ToolboxME for iSeries, MIDP, and IBM Toolbox for Java" on page 672

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Code example disclaimer

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Using ToolboxME for iSeries, MIDP, and JDBC

The following source illustrates one way that your ToolboxME for iSeries application can use the Mobile Information Device Profile (MIDP) and JDBC to access a database and store information offline.

This example demonstrates how a real estate agent might be able to view and bid on properties that are currently for sale. The agent uses a Tier0 device to access information for the properties, which is stored in the iSeries server database.

When built as a working program, the example code below connects to a database created for this purpose.

To create a working version of the source code and to obtain the source for creating and populating the required database, you must download the example. You also may want to review the instructions for creating and running the example program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME for iSeries example. This program is an example MIDlet that shows how
// you might code a JdbcMe application for the MIDP profile. Refer to the
// startApp, pauseApp, destroyApp and commandAction methods to see how it handles
// each requested transition.
//
////////////////////////////////////////////////////////////////////////////

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.sql.*;
import javax.microedition.rms.*;

import com.ibm.as400.micro.*;


public class JdbcMidpBid extends MIDlet implements CommandListener
```

```
{
    private static int BID_PROPERTY = 0;
    private Display   display;

    private TextField urlText = new TextField("urltext",
                                              "jdbc:as400://mySystem;user=myUid;password=myPwd;",
                                              65,
                                              TextField.ANY);
    private TextField jdbcmeText = new TextField("jdbcmetext", "meserver=myMEServer", 40, TextField.ANY);
    private TextField jdbcmeTraceText = new TextField("jdbcmetracetext", "0", 10, TextField.ANY);
    private final static String GETBIDS = "No bids are available, select here to download bids";
    private List      main = new List("JdbcMe Bid Demo", Choice.IMPLICIT);
    private List      listings = null;
    private Form      aboutBox;
    private Form      bidForm;
    private Form      settingsForm;
    private int       bidRow = 0;
    private String    bidTarget = null;
    private String    bidTargetKey = null;
    private TextField bidText = new TextField("bidtext", "", 10, TextField.NUMERIC);
    private Form      errorForm = null;

    private Command exitCommand   = new Command("Exit", Command.SCREEN, 0);
    private Command backCommand   = new Command("Back", Command.SCREEN, 0);
    private Command cancelCommand = new Command("Cancel", Command.SCREEN, 0);
    private Command goCommand     = new Command("Go", Command.SCREEN, 1);
    private Displayable   onErrorGoBackTo = null;

    /*
     * Construct a new JdbcMidpBid.
     */
    public JdbcMidpBid()
    {
        display = Display.getDisplay(this);
    }

    /**
     * Show the main screen
     */
    public void startApp()
    {
        main.append("Show Bids", null);
        main.append("Get New Bids", null);
        main.append("Settings", null);
        main.append("About", null);
        main.addCommand(exitCommand);
        main.setCommandListener(this);

        display.setCurrent(main);
    }

    public void commandAction(Command c, Displayable s)
    {
        // All exitCommand processing is the same.
        if (c == exitCommand)
        {
            destroyApp(false);
            notifyDestroyed();
            return;
        }
        if (s instanceof List)
        {
            List     current = (List)s;

            // An action occurred on the main page
            if (current == main)
            {
```

```
            int   idx = current.getSelectedIndex();
            switch (idx)
            {
            case 0:     // Show current bids
                showBids();
                break;
            case 1:     // Get New Bids
                getNewBids();
                break;
            case 2:     // Settings
                doSettings();
                break;
            case 3:     // About
                aboutBox();
                break;
            default :
                break;
            }
            return;
        } // current == main

        // An action occurred on the listings page
        if (current == listings)
        {
            if (c == backCommand)
            {
                display.setCurrent(main);
                return;
            }
            if (c == List.SELECT_COMMAND)
            {
                int idx = listings.getSelectedIndex();
                String stext = listings.getString(idx);
                if (stext.equals(GETBIDS))
                {
                    getNewBids();
                    return;
                }
                int   commaIdx = stext.indexOf(',');
                bidTargetKey = stext.substring(0, commaIdx);
                bidTarget = stext.substring(commaIdx+1) + "\n";
                // Also keep track of which offline result set row
                // This is. It happens to be the same as the index
                // in the list.
                bidRow = idx;

                bidOnProperty();
            }
        } // current == listings
        return;
    } // instanceof List
    if (s instanceof Form)
    {
        Form     current = (Form)s;
        if (current == errorForm)
        {
            if (c == backCommand)
                display.setCurrent(onErrorGoBackTo);

            return;
        } // errorForm
        if (current == settingsForm)
        {
            if (c == backCommand)
            {
                // Done with settings.
                display.setCurrent(main);
```

```
                settingsForm = null;
                return;
            }
        } // settingsForm
        if (current == aboutBox)
        {
            if (c == backCommand)
            {
                // Done with about box.
                display.setCurrent(main);
                aboutBox = null;
                return;
            }
        }
        if (current == bidForm)
        {
            if (c == cancelCommand)
            {
                display.setCurrent(listings);
                bidForm = null;
                return;
            }
            if (c == goCommand)
            {
                submitBid();
                if (display.getCurrent() != bidForm)
                {
                    // If we're no longer positioned at the
                    // bidForm, we will get rid of it.
                    bidForm = null;
                }
                return;
            }
            return;
        } // current == bidForm
    } // instanceof Form
}

public void aboutBox()
{
    aboutBox = new Form("aboutbox");
    aboutBox.setTitle("About");
    aboutBox.append(new StringItem("", "Midp RealEstate example for JdbcMe "));
    aboutBox.addCommand(backCommand);
    aboutBox.setCommandListener(this);
    display.setCurrent(aboutBox);
}

/**
 *  The settings form.
 */
public void doSettings()
{
    settingsForm = new Form("settingsform");
    settingsForm.setTitle("Settings");
    settingsForm.append(new StringItem("", "DB URL"));
    settingsForm.append(urlText);
    settingsForm.append(new StringItem("", "JdbcMe server"));
    settingsForm.append(jdbcmeText);
    settingsForm.append(new StringItem("", "Trace"));

    settingsForm.addCommand(backCommand);
    settingsForm.setCommandListener(this);
    display.setCurrent(settingsForm);
}
```

```
/**
 * Show the bid screen for the bid target
 * that we selected.
 */
public void bidOnProperty()
{
    StringItem item = new StringItem("", bidTarget);

    bidText = new TextField("bidtext", "", 10, TextField.NUMERIC);
    bidText.setString("");

    bidForm = new Form("bidform");
    bidForm.setTitle("Submit a bid for:");
    BID_PROPERTY = 0;
    bidForm.append(item);
    bidForm.append(new StringItem("", "Your bid:"));
    bidForm.append(bidText);
    bidForm.addCommand(cancelCommand);
    bidForm.addCommand(goCommand);
    bidForm.setCommandListener(this);
    display.setCurrent(bidForm);
}

/**
 * Update the listings card with the
 * current list of bids that we 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)
    {
    }
}
```

Example: Using ToolboxME for iSeries, MIDP, and IBM Toolbox for Java

The following source illustrates one way that your ToolboxME for iSeries application can use the Mobile Information Device Profile (MIDP) and IBM Toolbox for Java to access data and services on an iSeries server.

This example demonstrates each of the functions built into the IBM Toolbox for Java 2 Micro Edition support. This application features different pages or screens that illustrate some of the many ways in which your Tier0 device can use these functions.

When built as a working program, the example code below uses a Program Call Markup Language (PCML) file to run commands on the iSeries server.

To create a working version of the source code and to obtain the required PCML source for running server commands, you must download the example. You also may want to review the instructions for creating and running the example program.

Note: Read the Code example disclaimer for important legal information.

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME for iSeries example. This program is an example that shows how
```

```
// ToolboxME for iSeries can use PCML to access data and services on an
// iSeries server.
//
// This application requires that the qsyrusri.pcml file is present in the
// CLASSPATH of the MEServer.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.sql.*;
import java.util.Hashtable;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.*;

import com.ibm.as400.micro.*;


public class ToolboxMidpDemo extends MIDlet implements CommandListener
{
    private Display   display_;

    // A ToolboxME system object.
    private AS400 system_;

    private List      main_ = new List("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 iSeries.
                system_.connect();

                // Set the signon status text.
                signonStatusText_.setText("Signed on.");

                // Display a confirmation dialog that the user is signed on.
                feedback("Successfully signed on.", AlertType.INFO, main_);

                // Replace the SignOn button with SignOff.
                signonForm_.removeCommand(actionSignon_);
                signonForm_.addCommand(actionSignoff_);

                // Update the ticker.
                ticker_.setString("... Signed on to '" +
                                  signonSystemText_.getString() + "' as '" +
                                  signonUidText_.getString() + "' via '" +
                                  signonServerText_.getString() + "' ... ");
```

```
            }
            catch (Exception e)
            {
                e.printStackTrace();

                // Set the signon status text.
                signonStatusText_.setText(NOT_SIGNED_ON);

                feedback("Signon failed.  " + e.getMessage(), AlertType.ERROR);
            }
        }
        else if (action == actionSignoff_)
        {
            if (system_ == null)
                feedback("Internal error: System is null.", AlertType.ERROR);
            else
            {
                try
                {
                    // Disconnect from the iSeries.
                    system_.disconnect();
                    system_ = null;

                    // Set the signon status text.
                    signonStatusText_.setText(NOT_SIGNED_ON);

                    // Display a confirmation dialog that the user is no longer signed on.
                    feedback("Successfully signed off.", AlertType.INFO, main_);

                    // Replace the SignOff button with SignOn.
                    signonForm_.removeCommand(actionSignoff_);
                    signonForm_.addCommand(actionSignon_);

                    // Update the ticker.
                    ticker_.setString(NOT_SIGNED_ON);
                }
                catch (Exception e)
                {
                    feedback(e.toString(), AlertType.ERROR);

                    e.printStackTrace();

                    signonStatusText_.setText("Error.");

                    feedback("Error during signoff.", AlertType.ERROR);
                }
            }
        }
        else  // None of the above.
        {
            feedback("Internal error: Action is not recognized.", AlertType.INFO);
        }
    } // signonForm_
    else if (current == cmdcallForm_)
    {
        if (action == actionRun_)
        {
            // If the user has not signed on, display an alert.
            if (system_ == null)
            {
                feedback(NOT_SIGNED_ON, AlertType.ERROR);
                return;
            }

            // Get the command the user entered in the wireless device.
            String cmdString = cmdText_.getString();
```

```
            // If the command was not specified, display an alert.
            if (cmdString == null || cmdString.length() == 0)
                feedback("Specify command.", AlertType.ERROR);
            else
            {
                try
                {
                    // Run the command.
                    String[] messages = CommandCall.run(system_, cmdString);

                    StringBuffer status = new StringBuffer("Command completed with ");

                    // Check to see if their are any messages.
                    if (messages.length == 0)
                    {
                        status.append("no returned messages.");

                        cmdMsgText_.setText(null);

                        cmdStatusText_.setText("Command completed successfully.");
                    }
                    else
                    {
                        if (messages.length == 1)
                            status.append("1 returned message.");
                        else
                            status.append(messages.length + " returned messages.");

                        // If there are messages, display only the first message.
                        cmdMsgText_.setText(messages[0]);

                        cmdStatusText_.setText(status.toString());
                    }

                    repaint();
                }
                catch (Exception e)
                {
                    feedback(e.toString(), AlertType.ERROR);

                    e.printStackTrace();

                    feedback("Error when running command.", AlertType.ERROR);
                }
            }
        }
        else if (action == actionClear_)
        {
            // Clear the command text and messages.
            cmdText_.setString("");

            cmdMsgText_.setText(null);

            cmdStatusText_.setText(null);

            repaint();
        }
        else  // None of the above.
        {
            feedback("Internal error: Action is not recognized.", AlertType.INFO);
        }
    } // cmdcallForm_
    else if (current == pgmcallForm_)
    {
        if (action == actionRun_)
        {
            // If the user is not signed on before doing a program call, display an alert.
```

```
if (system_ == null)
{
    feedback(NOT_SIGNED_ON, AlertType.ERROR);
    return;
}

pgmMsgText_.setText(null);

// See the PCML example in the 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 iSeries if the Midlet is being destroyed or exited.
        if (system_ != null)
        {
            try
            {
                system_.disconnect();
            }
            catch (Exception e)
            {
            }
        }
    }
}
```

Examples: Utility classes

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java utility classes.

AS/400ToolboxJarMaker

- Example: Extracting AS400.class and all its dependent classes from jt400.jar
- Example: Splitting jt400.jar into a set of 300KB files
- Example: Removing unused files from a JAR file
- Example: Creating a 400KB smaller JAR file by omitting the conversion tables with the -ccsid parameter

CommandPrompter

- Example: Using CommandPrompter to prompt for and run a command

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Example: Using CommandPrompter

Note: Read the Code example disclaimer for important legal information.

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

Examples: Vaccess classes

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java vaccess classes.

AS400Panes

- Example: Creating an AS400DetailsPane to present the list of users defined on the systemAS400DetailsPane
- Example: Loading the contents of a details pane before adding it to a frame
- Example: Using an AS400ListPane to present a list of users
- Example: Using an AS400DetailsPane to display messages returned from a command call
- Example: Using an AS400TreePane to display a tree view of a directory
- Example: Using an AS400ExplorerPane to present various print resources

Command call

- Example: Creating a CommandCallButton
- Example: Adding the ActionCompletedListener to process all iSeries messages that a command generates
- Example: Using the CommandCallMenuItem

Data queues

- Example: Creating a DataQueueDocument
- Example: Using a DataQueueDocument

Error events

- Example: Handling error events
- Example: Defining an error listener
- Example: Using a customized handler to handle error events

Integrated file system

- Example: Using IFSFileDialog
- Example: Using IFSFileSystemView
- Example: Using IFSTextFileDocument

JDBC
- Example: Using the JDBC driver to create and populate a table
- Example: Using the JDBC driver to query a table and output its contents
- Example: Creating an AS400JDBCDataSourcePane

Jobs
- Example: Creating a VJobList and presenting the list in an AS400ExplorerPane
- Example: Presenting a list of jobs in an explorer pane

Messages
- Example: Using VMessageQueue

Program call
- Example: Creating a ProgramCallMenuItem
- Example: Processing all program generated iSeries messages
- Example: Adding two parameters
- Example: Using a ProgramCallButton in an application

Print
- Example: Using VPrinter
- Example: VPrinterOutput

Record-level access
- Example: Creating a RecordListTablePane object to display all records less than or equal to a key
- Example: Using RecordListFormPane

SpooledFileViewer
- Example: Creating a Spooled File Viewer to view a spooled file previously created on an iSeries

SQL
- Example: Using SQLQueryBuilderPane
- Example: Using SQLResultSetTablePane

System values
- Example: Creating a system value GUI using the AS400Explorer pane

Users and groups
- Example: Creating a VUserList with the AS400DetailsPane
- Example: Using an AS400ListPane to create a list of users for selection

The following disclaimer applies to all of the IBM Toolbox for Java examples:

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Example: Using VUserList

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////
//
// 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]);
        }
    }


}
```

Example: Using VMessageList

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// VMessageList example.  This program presents a details
// view of messages returned from a command call.
//
// Command syntax:
//    VMessageListExample system
//
// This source is an example of IBM Toolbox for Java "VMessageList".
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VMessageListExample
{


    public static void main (String[] args)
    {
```

```
        // If a system was not specified, then display help text and
        // exit.
        if (args.length != 1)
        {
            System.out.println("Usage:  VMessageListExample system");
            return;
        }

        try
        {
            // Create an AS400 object.  The system name was passed
            // as the first command line argument.
            AS400 system = new AS400 (args[0]);

            // Create a CommandCall object a run the command.
            CommandCall command = new CommandCall (system);
            command.run ("CRTLIB FRED");

            // Create a VMessageList object with the messages
            // returned from the command call.
            VMessageList messageList = new VMessageList (command.getMessageList ());

            // Create a frame.
            JFrame f = new JFrame ("VMessageList example");

            // Create an error dialog adapter.  This will display
            // any errors to the user.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Create a details pane to display the message list.
            // Use load to load the information.
            AS400DetailsPane detailsPane = new AS400DetailsPane (messageList);
            detailsPane.addErrorListener (errorHandler);
            detailsPane.load ();

            // When the frame closes, exit.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Layout the frame with the details pane.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", detailsPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Example: Using VIFSDirectory

Note: Read the Code example disclaimer for important legal information.

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


}
```

Example: Using VPrinters

Note: Read the Code example disclaimer for important legal information.

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


}
```

Example: Using CommandCallMenuItem

Note: Read the Code example disclaimer for important legal information.

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

```
    }

}
```

Example: Using DataQueueDocument

Note: Read the Code example disclaimer for important legal information.

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

Example: Using IFSFileDialog

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////////
//
// 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 = iSeries server");
         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);
   }
}
```

Example: Using IFSTextFileDocument

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////
//
// IFS text file document example.  This program demonstrates how to
// use a document that is associated with a text file in the AS/400
// 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

The AS400JDBCDataSourcePane class presents the property values of an AS400JDBCDataSource object. Optionally, changes can be made to the AS400JDBCDataSource object.

AS400JDBCDataSourcePane extends JComponent. To use an AS400JDBCDataSourcePane to display the properties of a data source, the data source can either be specified on the AS400JDBCDataSourcePane constructor or set after the AS400JDBCDataSourcePane has been created using setDataSource(). Changes made to the graphical user interface (GUI) can be applied to the data source object using applyChanges().

Example: Using AS400JDBCDataSourcePane

The following example creates an AS400JDBCDataSourcePane and an **OK** button and adds them to a frame. Changes made to the GUI are applied to the data source when you click **OK**.

```
// 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 ();
```

Example: Using VJobList to present a list of jobs

Note: Read the Code example disclaimer for important legal information.

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


}
```

Example: Using VMessageQueue

Note: Read the Code example disclaimer for important legal information.

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


}
```

Example: Using a button to call a program on the server

Note: Read the Code example disclaimer for important legal information.

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


}
```

Example: Using VPrinter

Note: Read the Code example disclaimer for important legal information.

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

Example: Using VPrinters

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////
//
// 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 Example

Note: Read the Code example disclaimer for important legal information.

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

Example: Using SQLQueryBuilderPane

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////
//
// SQLQueryBuilderPane example.  This program presents a query builder
// which 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 ();
    }


}
```

Example: Using SQLResultSetTablePane

Note: Read the Code example disclaimer for important legal information.

```
///////////////////////////////////////////////////////////////////////
//
// SQLResultSetTablePane example.  This program presents the contents of
// a table in a table pane.  There is a SQLStatementDocument which allows
// the user to type in any SQL statement.  In addition, there is a button
```

```
// which 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);
        }
    }


}
```

Examples: XPCML

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java XPCML component.

- "Example: Condensing an existing XPCML document" on page 721
- "Example: Condensing an existing XPCML document" on page 721
- "Example: Using condensed XPCML to create a ProgramCallDocument object" on page 724
- "Example: Obtaining program call results as condensed XPCML" on page 724
- "Example: Retrieving program call results as XPCML"
- "Example: Passing in parameter values as XPCML" on page 718
- "Examples: Passing in arrays of parameter values as XPCML" on page 719
- "Example: Converting a PCML document to an XPCML document" on page 417

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Example: Retrieving program call results as XPCML

The following example shows how you can construct an XPCML ProgramCallDocument, call an iSeries program, and retrieve the results of the program call as XPCML. The example supposes the following components:

- XPCML document qgyolaus.xpcml, which defines the program and parameter specifications with input values
- Java code that constructs a ProgramCallDocument object, uses the XPCML file, and then calls program QGYOLAUS
- Results of the program call, which the Java code generates as XPCML and stores in file XPCMLOut.xpcml

Notice how array data is specified in the original and the generated XPCML. The element qgyolaus.receiver, an output parameter, is an XPCML arrayOfStructParm with an attribute that sets the count to listInfo.rcdsReturned. The following example code includes only part of the QGYOLAUS output. If the example included all the output, the code might list 89 users under the <arrayOfStructParm> XPCML tag.

For arrays of structs, XPCML uses the <struct_i> XPCML tag to delimit each structParm element. Each <struct_i> tag indicates that the data enclosed within it is one element of type autu0150 struct. The index attribute of the <struct_i> tag specifies the element of the array for the struct.

For arrays of simple types, such as arrayOfStringParm, arrayOfIntParm, and so on, the <i> XPCML tag lists array elements.

XPCML document 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>
```

Java code that constructs the ProgramCallDocument object and uses the XPCML to call program QGYOLAUS

```
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");
```

Results of the program call, generated as XPCML and stored in file XPCMLOut.xpcml

```
<?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>
          <unsignednIntParm 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>
```

Example: Passing in parameter values as XPCML

Program parameter values can be set in the XPCML source file. When the XPCML is read in and parsed, the ProgramCallDocument setValue method is called automatically for each parameter whose value has been passed in as XPCML. This relieves the user from having to write Java code to set the values of complicated structures and arrays.

In the following examples, the XPCML calls two different programs, prog1 and prog2. Both programs use the input parameter s1Ref. The first example sets different values for s1Ref for each program call. The second example specifies the same value for s1Ref for each program call, which illustrates a useful way to set constant data values for input parameters.

Example: Passing in different values for input paramters

In the following example, after the XML parser reads in and parses the document, the value of element prog1.s1Ref.s2Ref.s2p1[0] is prog1Val_1 and the value of element prog1.s1Ref.s2Ref.s2p1[1] is prog1Val_2.

```
    <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>
```

Example: Passing in constant values for input paramters

In the following example, after the XML parser reads in and parses the document, the value of element prog1.s1Ref.s2Ref.s2p1[0] is constantVal_1 and the value of element prog1.s1Ref.s2Ref.s2p1[1] is constantVal_2.

```
<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>
```

Examples: Passing in arrays of parameter values as XPCML

When using XPCML to pass in array data, you must use the count attribute:

- Specify the count attribute on the array element
- Set the count attribute to the number of elements that the array contains at the time you parse the document

The following example illustrates how to pass in arrays of parameter values by using structParm array data and an array of structs.

```
<?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>
```

For example, the following XPCML specifies an array of 3 intParms and sets the first element to 12, the second to 100, and the third to 4:

```
<?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>
```

Using the index attribute of the <i> and <struct_i> tags to set array values

You can use the index attribute of the <i> and <struct_i> tags to help you set array values. In the following example, the XPCML sets the first element of the array to 4, the second to 100, and the third to 12.

```
<?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>
```

Example: Condensing an existing XPCML document

The following example illustrates how to condense an existing XPCML document. The example includes original XPCML source, the resulting condensed XPCML, and the extended schema.

Original XPCML source

```
<?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>
```

Condensed XPCML source

```
<?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>
```

Generated schema

```
<!-- 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>
```

Example: Condensing an existing XPCML document, including Java code

The following example illustrates how to condense an existing XPCML document. The example includes original XPCML source, the resulting condensed XPCML, the Java code that calls condenseXPCML(), and a few of the newly generated type definitions in the extended schema:

Original XPCML source

```
<!-- 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>
```

Java code to condense the original XPCML source

```
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();
}
```

Condensed XPCML source: 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>
```

Some type definitions from the generated schema: myXSD.xsd

```
<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>
```

Example: Using condensed XPCML to create a ProgramCallDocument object

Some ProgramCallDocument constructors accept a condensedXPCML source file and corresponding schema (.xsd file). This enables you to use condensed XPCML to create a ProgramCallDocument object.

The previously mentioned constructors require that you provide the following parameters:

- A String that specifies a condensed XPCML file
- An InputStream that contains the type definitions created by running condenseXPCML()

Using these constructors loads and parses a condensed XPCML file. Additionally, the process logs any parse errors. After completing the parse, the constructor creates a ProgramCallDocument object.

The following Java code example uses condensed XPCML to create a ProgramCallDocument object. The example code assumes the following

- The name of the condensed XPCML file is myCondensedXPCML.xpcml
- The name of the extended schema is myXSD.xsd

The code then uses the ProgramCallDocument object to run program qusrjobi_jobi0100.

```
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");
```

Note: The XPCML code that you use to call the program (after creating the ProgramCallDocument object) is the same as the code you would use with PCML.

Example: Obtaining program call results as condensed XPCML

You use the same process to obtain the results of a program call as condensed XPCML or noncondensed XPCML. All you need to do is call ProgramCallDocument.generateXPCML().

Use setXsdName() to specify the name of the extended schema, which generateXPCML() uses to generate the noNamespaceSchemaLocation attribute of the <xpcml> tag in the condensed XPCML.

Using setXsdName() is important when you want to use the program call results (in condensed XPCML) as source for another ProgramCallDocument object. You must specify the name of the extended schema so that the parser knows which schema file to use when parsing.

For example, the following code obtains the results from a program call and generates condensed XPCML.

```
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");
}
```

The following code shows an example of obtaining program call results as condensed XPCML:

```
<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>
```

Related information for IBM Toolbox for Java

The following list includes Web sites and Information Center topics that relate to the IBM Toolbox for Java information.

IBM Toolbox for Java resources

Use the following sites to learn more about the IBM Toolbox for Java:

- IBM Toolbox for Java and JTOpen  : Offers information about service packs, performance tips, examples, and much more. You can also download a zipped package of this information, including the javadocs.

- IBM Toolbox for Java Frequently Asked Questions (FAQ) : Provides answers to questions about performance, troubleshooting, JDBC, and more.
- IBM Toolbox for Java and JTOpen forum : Offers an effective way to communicate with the community of Java programmers who use IBM Toolbox for Java and the IBM Toolbox for Java developers themselves.

IBM Toolbox for Java 2 Micro Edition resources

Use the following sites to learn more about ToolboxME for iSeries and the Java implementation of wireless technologies:

- IBM Toolbox for Java and JTOpen : Offers more information about ToolboxME for iSeries.
- IBM alphaWorks® Wireless : Offers information about new wireless technologies, including downloads and links to development resources.
- Sun Java 2 Platform, Micro Edition : Provides additional information about the Java wireless technologies, including the following:
  - K Virtual Machine (KVM)
  - Connected Limited Device Configuration (CLDC)
  - Mobile Information Device Profile (MIDP)
- Java Wireless Developer : Offers a wide range of technical information for Java wireless application developers.
- Wireless application development tools:
  - IBM WebSphere Studio Device Developer
  - Java 2 Platform Micro Edition, Wireless Toolkit

Java

Java is a programming language that allows you to develop portable object-oriented applications and applets. Use the following sites to learn more about Java:

- IBM developerWorks® Java technology zone : Offers information, education, and tool to help you use Java, IBM products, and other technologies to create business solutions.
- IBM alphaWorks Java : Offers information about new Java technologies, including downloads and links to development resources.
- "The Source for Java Technology" from Sun Microsystems : Offers information about the various uses for Java, including new technologies.

Java Naming and Directory Interface

- Java Naming and Directory Interface(TM) (JNDI) : Offers an overview of JNDI, technical information, examples, and a list of available service providers.
- iSeries Directory Server (LDAP) : Provides information about LDAP (Lightweight Directory Access Protocol) on i5/OS.

Java Secure Socket Extension

- Java Secure Socket Extension (JSSE) : Offers a brief overview of JSSE and links to more information.

Servlets

Servlets are small Java programs that run on a server and mediate requests from one or many clients (each of which runs in a browser) to one or more databases. Because servlets are programmed in Java, they can execute requests as multiple threads within a single process, thereby saving system resources. Use the following sites to learn more about servlets:

- IBM Websphere, IBM PartnerWorld® : Offers information about the servlet-based Web application server.
- Java Servlet technology : Provides technical information, instructions, and tools to help you understand and use servlets.

XHTML

XHTML is touted as the successor to HTML 4.0. It is based on HTML 4.0, but incorporates the extensibility of XML. Use following sites to learn more about XHTML:

- The Web Developer's Virtual Library : Offers an introduction to XHTML, including examples and links to additional information.
- W3C : Provides technical information about XHTML standards and recommendations.

XML

Extensible Markup Language (XML) is a metalanguage that allows you to describe and organize information in ways that are easily understandable by both humans and computers. A metalanguage allows you to define a document markup language and its structure. Use the following sites to learn more about XML:

- IBM developerWorks XML zone : Provides a site dedicated to the work IBM does with XML and how it works to facilitate e-commerce
- IBM alphaWorks XML : Offers information about emerging XML standards and tools, including downloads and links to development resources.
- W3C XML : Offers technical resources for XML developers.
- XML.com : Offers updated information about XML in the computer industry
- XML.org : Provides news and information about the XML community, including industry news, calendars of events, and more.
- XML Cover Pages : Provides a comprehensive online reference work for XML, SGML, and related XML standards, like XSL and XSLT.

Other references

- IBM HTTP Server for iSeries : Provides information, resources, and tips about the IBM HTTP Server for iSeries.
- iSeries Access for Windows : Offers information about iSeries Access for Windows, including downloads, FAQs, and links to additional sites.
- IBM WebSphere Host On-Demand : Provides information about the browser-based emulator that offers support for S/390®, iSeries, and DEC/Unix emulation.
- IBM Support and downloads : Offers a portal to IBM hardware and software support.

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This IBM Toolbox for Java publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM Toolbox for Java.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

| Advanced Function Presentation
| Advanced Function Printing
| AFP
| AIX
| alphaWorks
| AS/400
| DB2
| DB2 Universal Database
| developerWorks
| i5/OS
| IBM
| IPDS
| iSeries
| NetServer
| OS/400
| PartnerWorld
| S/390
| SecureWay
| VisualAge
| WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

| Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Printed in USA