- Clases HTML
  - Clase BidiOrdering
  - Clase HTMLAlign
  - Clases de formularios HTML
    - Clases de entrada de formulario
      - ButtonFormInput
      - FileFormInput
      - HiddenFormInput
      - ImageFormInput
      - ResetFormInput
      - SubmitFormInput
      - TextFormInput
      - PasswordFormInput
      - RadioFormInput
      - CheckboxFormInput
    - Clases LayoutFormPanel
      - GridLayoutFormPanel
      - LineLayoutFormPanel
    - Clase TextAreaFormElement
    - Clase LabelFormElement
    - Clase SelectFormElement
    - Clase SelectOption
    - Clase RadioFormInputGroup
  - Clase HTMLHeading
  - Clase HTMLHyperlink
  - HTMLImage
  - Clases HTMLList
  - Clase HTMLMeta
  - Clase HTMLParameter
  - Clase HTMLServlet
  - Clases de tablas HTML
    - Clase HTMLTableCell
    - Clase HTMLTableRow
    - Clase HTMLTableHeader
    - Clase HTMLTableCaption
  - Clase HTMLText
  - Clases HTMLTree
    - Clase HTMLTreeElement

IBM Toolbox para Java

IBM Toolbox para Java es un conjunto de clases Java(TM) que le permiten utilizar programas Java para acceder a los datos de los servidores iSeries y AS/400e. Con estas clases, puede escribir aplicaciones de cliente/servidor, applets y servlets que funcionen con datos existentes en el iSeries. También puede ejecutar las aplicaciones Java que utilizan las clases de IBM Toolbox para Java en la máquina virtual Java (JVM) de iSeries.

IBM Toolbox para Java utiliza los servidores de sistema principal de iSeries como puntos de acceso al sistema.Dado que Toolbox para Java utiliza las funciones de comunicaciones incorporadas en Java, no es necesario utilizar IBM iSeries Access Express para Windows a fin de emplear Toolbox para Java. Cada servidor se ejecuta en un trabajo aparte en el servidor, y cada trabajo servidor envía y recibe corrientes de datos a través de una conexión por socket.

Puede obtener más información acerca de IBM Toolbox para Java mediante la barra de navegación principal o los enlaces siguientes:

Novedades de V5R2
Conozca los cambios significativos realizados, las funciones mejoradas y otros aspectos de interés.

Imprimir el tema de IBM Toolbox para Java
Vea o baje un PDF del tema de Toolbox para Java. También puede bajar el tema de Toolbox para Java en un paquete comprimido.

Utilizar la clase finder
Realice búsquedas de clases por nombre y descripción de forma fácil y rápida, consulte clases por paquete o examine una lista alfabética de todas las clases de Toolbox para Java.

Guía de iniciación a IBM Toolbox para Java
Descubra cómo gestionar la instalación de IBM Toolbox para Java. Aprenda a efectuar su instalación en estaciones de trabajo y servidores. Utilice los ejemplos simples de programación para ver cómo puede empezar a utilizar las clases de Toolbox para Java en las aplicaciones.

Clases de IBM Toolbox para Java
Consulte las diversas clases de los paquetes de IBM Toolbox para Java que le permiten trabajar con los datos de los servidores iSeries y AS/400e. Esta información contiene descripciones, código de ejemplo y datos técnicos que le ayudarán a crear programas de IBM Toolbox para Java.

Utilización de la Caja de Herramientas Gráfica para crear sus propios paneles de interfaz gráfica de usuario (GUI)
Utilice la Caja de Herramientas Gráfica para crear paneles de interfaz de usuario personalizados en Java, que puede incorporar a aplicaciones Java, applets o conectores de iSeries Navigator.

JavaBeans
Consulte cómo crear JavaBeans mediante las clases públicas de Toolbox para Java, creadas según los estándares JavaBean de Javasoft. Examine los ejemplos que muestran cómo utilizar los JavaBeans en los programas.

JDBC
Descubra el soporte para JDBC que ofrece Toolbox para Java. Con JDBC, los programas pueden emitir sentencias SQL (Structured Query Language) a las bases de datos de los servidores y procesar los resultados obtenidos de ellas.

Utilización de PCML para efectuar llamadas a programas de iSeries
El lenguaje PCML (Program Call Markup Language) le ayuda a efectuar llamadas a programas de iSeries utilizando menos código Java. PCML es una sintaxis de códigos, basada en XML, que describe por completo los parámetros de entrada y salida para los programas de iSeries a los que llama la aplicación Java.

Soporte de proxy
Averigüe cómo se utiliza el soporte de proxy de IBM Toolbox para Java, que incluye el uso del protocolo SSL (capa de sockets segura) para cifrar datos.

»Utilización de RFML para definir y gestionar formatos de datos
Utilice RFML (Record Format Markup Language) para separar las especificaciones de formato de datos de la lógica de empresa de los programas Java. RFML es una sintaxis de códigos, basada en XML y con una estrecha relación con PCML, que permite a las aplicaciones Java especificar y manipular campos dentro de determinados tipos de registros.«

»Seguridad
Infórmese acerca de cómo utilizar JSSE (Java Secure Socket Extension) y Toolbox para Java para proporcionar un intercambio de datos seguro entre los clientes y servidores que ejecutan cualquier protocolo de aplicaciones a través de TCP/IP.«

»Depurador del sistema iSeries
Utilice la interfaz gráfica de usuario (GUI) del depurador del sistema iSeries para depurar y probar los programas que se ejecutan en el servidor iSeries.«

Propiedades del sistema
Aprenda a emplear las propiedades del sistema para configurar diversos aspectos de IBM Toolbox para Java, tales como la definición de un servidor proxy o un nivel de rastreo. Puede utilizar las propiedades del sistema para llevar a cabo una práctica configuración en tiempo de ejecución sin volver a compilar el código.

»IBM Toolbox para Java 2 Micro Edition
Utilice este nuevo componente de Toolbox para Java para escribir programas Java para una amplia gama de dispositivos inalámbricos. Con ToolboxME para iSeries, los dispositivos inalámbricos pueden acceder directamente a los datos y recursos del servidor iSeries.«

Javadocs de IBM Toolbox para Java
Vea la información de consulta de javadocs que hace referencia a las clases de IBM Toolbox para Java.

»Preguntas habituales (FAQ)
Encuentre la respuesta a cuestiones relacionadas con la optimización del rendimiento de IBM Toolbox para Java, la resolución de problemas, la utilización de JDBC y muchos temas más. «

Entre la información adicional se encuentra la siguiente:

- Una lista de ejemplos de programación de Toolbox para Java
- Consejos de programación para ayudarle a utilizar Toolbox para Java
- Información relacionada, con enlaces a más información sobre Java, servlets, XML y otros temas.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

Novedades de V5R2

IBM Toolbox para Java está disponible en los formatos siguientes:

- El programa bajo licencia de IBM Toolbox para Java, 5722-JC1, Versión 5 Release 2 (V5R2) se instala en las versiones V4R5 y posteriores de OS/400. Desde un cliente, IBM Toolbox para Java se conecta a versiones V4R5 y posteriores de OS/400.
- OS/400 también incluye las clases no gráficas de IBM Toolbox para Java optimizadas para el uso al ejecutar las clases de IBM Toolbox para Java en la máquina virtual Java (JVM) de iSeries. Por consiguiente si, por ejemplo, no necesita las funciones gráficas del programa bajo licencia, puede seguir utilizando fácilmente IBM Toolbox para Java. Encontrará más información en Archivos Jar.
- IBM Toolbox para Java ahora también incorpora fuente abierto. Puede bajar el código y obtener información del sitio Web de JTOpen.

Paquetes nuevos

IBM Toolbox para Java 2 Micro Edition es un paquete nuevo de IBM Toolbox para Java. El nuevo paquete com.ibm.as400.micro proporciona un conjunto de clases que permiten escribir programas Java que se ejecutan en dispositivos inalámbricos tales como organizadores personales y teléfonos móviles. Debe bajar por separado el paquete micro.

El depurador del sistema iSeries dota a la estación de trabajo de un nuevo entorno gráfico de depuración para los programas ILE, Java, C y C++ que se ejecutan en un servidor iSeries.

Clases nuevas

IBM Toolbox para Java V5R2 también incluye numerosas clases nuevas en los paquetes ya existentes. Las clases nuevas le permiten realizar estas tareas:

- Utilizar las clases ClusteredHashTable para compartir y duplicar datos no persistentes entre los nodos de un clúster.
- Utilizar la clase CommandPrompter para solicitar parámetros en un mandato determinado.
- Para obtener información acerca de las clases JDBC(TM) nuevas, consulte Clases JDBC nuevas y funciones mejoradas.
- La clase RecordFormatDocument permite utilizar el nuevo componente RFML (Record Format Markup Language) para especificar formatos de registro y crear, leer y escribir registros de datos.

Clases mejoradas

IBM Toolbox para Java V5R2 también contiene mejoras en clases ya existentes. Estas mejoras ofrecen lo siguiente:

- Adición del soporte de kerberos a los objetos AS400, que ahora utilizan la infraestructura JGSS (Java Generic Security Service) a fin de llevar a cabo la autenticación para los servidores de Toolbox para Java.
- Los objetos SecureAS400 ahora pueden emplear la infraestructura JSSE (Java Secure Socket Extension) para cifrar los datos que fluyen entre el cliente y el servidor.
- Cambios en los códigos y las funciones de PCML (Program Call Markup Language):
  - Diversos atributos nuevos del código <data> añaden soporte para las series Unicode y permiten a los usuarios especificar cómo deben eliminarse los espacios de las series.
  - Diversos atributos nuevos y modificados del código <program> ahora permiten que se especifique la vía de acceso en tiempo de ejecución y admiten la especificación de un CCSID para el nombre de punto de entrada de programa de servicio.

- Para el rastreo, PCML ahora requiere el uso de la clase Trace (com.ibm.as400.access.Trace) en lugar de la clase PcmlMessageLog.

Clases JDBC nuevas y funciones mejoradas

El soporte para JDBC de IBM Toolbox para Java V5R2 incluye clases nuevas y funciones mejoradas, tales como el soporte para la interfaz de programación de aplicaciones (API) JDBC 3.0. Algunos de los importantes cambios efectuados son:

- AS400JDBCSavepoint es una clase nueva (con soporte para JDBC 3.0) que hace posible un control más preciso de las retrotracciones de las transacciones.
- AS400JDBCParameterMetaData es una clase nueva (con soporte para JDBC 3.0) que permite recuperar los tipos y las propiedades de los parámetros de los objetos PreparedStatement y CallableStatement.
- Se ha añadido la posibilidad de conectarse a agrupaciones de almacenamiento auxiliar independiente (IASP).
- Varios métodos nuevos de objeto binario de gran tamaño (blob) y objeto de tipo carácter de gran tamaño (clob) (con soporte para JDBC 3.0) permiten actualizar los valores de estos tipos de datos.
- Una propiedad de JDBC nueva (extended metadata) mejora la función de reportar atributos de ResultSetMetaData.
- Se han efectuado otras mejoras que mejoran el soporte para JDBC.

Componente XML nuevo

En la versión V5R2, IBM Toolbox para Java ha incorporado RFML (Record Format Markup Language), una extensión de XML parecida a PCML. RFML permite utilizar XML en los programas Java a fin de especificar el formato de los almacenamientos intermedios de datos y los formatos de registro de archivo físico, así como examinar el contenido de los almacenamientos intermedios y registros recuperados.

Funciones y características adicionales de la Caja de Herramientas Gráfica

La Caja de Herramientas Gráfica incorpora nuevas características:

- Visualización de casillas en una columna de tabla como recuadros de selección
- Especificación de una altura y una anchura mínimas para los diálogos a fin de garantizar la visualización correcta de los elementos de los diálogos
- Especificación de que la primera columna de una tabla contiene un árbol jerárquico dinámico, donde cada casilla corresponde a un nodo del árbol

Para obtener más información sobre estas funciones, consulte la ayuda en línea del Constructor de GUI.

Compatibilidad

IBM Toolbox para Java ya no soporta la ejecución en la máquina virtual Java por omisión en Netscape(R) Navigator o Microsoft(R) Internet Explorer. Para que el applet que utiliza las clases de Toolbox para Java se ejecute en un navegador, debe instalar un conector como el conector de Sun Java 2 Runtime Environment (JRE) 1.3.0  .

Toolbox para Java ya no incluye data400.jar. Las clases que contenía data400.jar ahora se encuentran en jt400.jar. Elimine data400.jar de las sentencias CLASSPATH.

Los métodos getObject() para ResultSet y CallableStatement ahora devuelven objetos Integer cuando SQLType es SMALLINT. Estos métodos antes devolvían objetos Short. Si utiliza readObject para leer columnas SMALLINT,

debe modificar la aplicación Java para dar cabida al nuevo tipo de objeto devuelto.

Varios modos de reportar errores al lanzar errores de truncamiento de datos producen avisos que no hacen que falle la aplicación.

No puede utilizar este release de IBM Toolbox para Java a fin de deserializar algunos objetos serializados con releases anteriores a V5R1.

Si utiliza SSL (capa de sockets segura) para cifrar los datos que fluyen entre el cliente y el servidor, debe emplear uno de los elementos siguientes:

- JSSE (Java Secure Socket Extension)
- Los objetos SSL proporcionados en una versión V5R1 o posterior del programa bajo licencia IBM iSeries Client Encryption 5722-CE2 o 5722-CE3. Este release de IBM Toolbox para Java no funciona con las versiones V4R5 y anteriores de iSeries Client Encryption.

IBM Toolbox para Java sigue proporcionando soporte para:

- Swing 1.1, que es obligatorio para utilizar las clases de GUI o la Caja de Herramientas Gráfica
- Plataforma Java 2, Standard Edition (J2SE), con soporte continuado de Java Development Kit 1.1.8

Se recomienda consultar asimismo los requisitos de OS/400 para ejecutar IBM Toolbox para Java.

Novedades hasta la fecha 26 de septiembre de 2002

Clase finder de IBM Toolbox para Java
Utilice la nueva clase finder para realizar búsquedas de clases por nombre y descripción de forma fácil y rápida, consultar clases por paquete o examinar una lista alfabética de todas las clases de IBM Toolbox para Java. La descripción breve de cada clase es un enlace a información más específica.

Cómo ver las novedades y los cambios realizados

Para ayudarle a ver dónde se han realizado cambios técnicos, esta información (aunque no los javadocs) utiliza los símbolos siguientes:

- » marca el principio de información nueva o cambiada
- « marca el final de información nueva o cambiada

Si desea obtener otra información sobre las novedades o modificaciones de este release, consulte el documento Memo to Users.

Imprimir este tema

Para ver o bajar la versión PDF, seleccione PDF IBM Toolbox para Java (alrededor de 5,4 MB u 811 páginas).

Nota: parte de la información contenida en el tema de IBM Toolbox para Java no está incluida en los archivos PDF.

Guardar archivos PDF

Para guardar un PDF en la estación de trabajo con el fin de verlo o imprimirlo:

1. Pulse con el botón derecho en el PDF en el navegador (pulse con el botón derecho en el enlace situado más arriba).
2. Pulse **Guardar destino como**.
3. Navegue al directorio en el que desea guardar el archivo PDF.
4. Pulse **Guardar**.

Bajar Adobe Acrobat Reader

Si necesita Adobe Acrobat Reader para ver o imprimir estos PDF, puede bajar una copia del sitio Web de Adobe (www.adobe.com/products/acrobat/readstep.html) .

Para guardar un PDF en la estación de trabajo con el fin de verlo o imprimirlo:

1. Abra el PDF en el navegador (pulse el enlace situado más arriba).
2. En el menú del navegador, pulse **Archivo**.
3. Pulse **Guardar como**.
4. Navegue al directorio en el que desea guardar el archivo PDF.
5. Pulse **Guardar**.

Bajar información de IBM Toolbox para Java en un paquete comprimido

Puede bajar un paquete comprimido del tema de IBM Toolbox para Java que incluye los javadocs en el sitio Web de IBM Toolbox para Java y JTOpen .

Nota: la información de **IBM Toolbox para Java** tiene enlaces con documentos que no se incluyen en el paquete comprimido. Dichos enlaces no funcionarán en los archivos que baje a la estación de trabajo.

Guía de iniciación a IBM Toolbox para Java

El uso de IBM Toolbox para Java facilita la tarea de escribir applets, servlets y aplicaciones Java de cliente que accedan a los recursos, datos y programas de iSeries.

La información siguiente le ayudará a instalar y empezar a utilizar IBM Toolbox para Java:

»Gestión de la instalación
Consulte cómo las distintas formas de instalar y gestionar la instalación de Toolbox para Java afectan a la facilidad de gestión y al rendimiento. «

Instalación de Toolbox para Java
Lea los requisitos de OS/400 y la estación de trabajo para instalar Toolbox para Java en un entorno de cliente/servidor. Aprenda a instalar Toolbox para Java en las estaciones de trabajo y los servidores iSeries.

Propiedades del sistema
Obtenga información sobre las propiedades del sistema y aprenda cómo puede emplearlas para configurar diversos aspectos de IBM Toolbox para Java.

»Ejemplos de programación simples
Adéntrese en el uso de Toolbox para Java. Cree su primer programa de Toolbox para Java o examine los ejemplos de programación simples que se facilitan. Los ejemplos muestran cómo empezar a utilizar Toolbox para Java a fin de trabajar con los datos y servicios disponibles en el servidor iSeries. «

Gestión de la instalación de IBM Toolbox para Java

Sólo es necesario que instale IBM Toolbox para Java en los sistemas clientes que lo utilicen o en una ubicación de la red en la que los clientes puedan acceder al mismo. Los clientes pueden ser PC, estaciones de trabajo dedicadas o sistemas iSeries. Es importante recordar que puede configurar un servidor iSeries o una partición del servidor como cliente. En el segundo caso, debe instalar Toolbox para Java en la partición cliente del servidor.

Puede emplear cualquiera de los métodos siguientes (solos o combinados) para instalar y gestionar Toolbox para Java:

- Gestión individual para instalar y gestionar de forma individual Toolbox para Java en cada cliente
- Gestión en red de las instalaciones de cliente utilizando AS400ToolboxInstaller para instalar y gestionar Toolbox para Java en cada cliente
- Gestión en red de una sola instalación utilizando la red para instalar y gestionar una única instalación compartida de Toolbox para Java en un servidor

En los apartados siguientes se describe brevemente cómo afecta al rendimiento y a la facilidad de gestión cada uno de los métodos. El modo de desarrollo de las aplicaciones Java y gestión de los recursos que elija determinará cuál de los métodos (o cuál de las combinaciones de métodos) utilizará.

Gestión individual

Puede elegir gestionar las instalaciones de Toolbox para Java de forma individual en los distintos clientes. La principal ventaja de instalar Toolbox para Java en clientes individuales es que con ello se reduce el tiempo que tarda un cliente en iniciar una aplicación que utilice las clases de Toolbox para Java.

La principal desventaja es la gestión individual de esas instalaciones. Un usuario o una aplicación creada por el usuario debe hacer un seguimiento de qué versión de Toolbox para Java está instalada en cada estación de trabajo y llevar a cabo las tareas de gestión.

Gestión en red de las instalaciones de cliente

Puede elegir utilizar la red y AS400ToolboxInstaller para gestionar las instalaciones de cliente de Toolbox para Java. Como cada cliente tiene su propia copia de Toolbox para Java, este tipo de instalación tiene la misma ventaja de reducción del tiempo que tarda un cliente en iniciar una aplicación de Toolbox para Java. Asimismo, permite actualizar automáticamente todas las instalaciones individuales de Toolbox para Java.

La principal desventaja de este método radica en la creación y el mantenimiento del proceso que utiliza AS400ToolboxInstaller para gestionar las instalaciones individuales.

Gestión en red de una sola instalación

También puede utilizar la red para instalar y gestionar una única copia de Toolbox para Java en un servidor al que todos los clientes puedan acceder. Este tipo de instalación en red proporciona las siguientes ventajas:

- Todos los clientes utilizan la misma versión de IBM Toolbox para Java.
- La actualización de la única instalación de Toolbox para Java beneficia a todos los clientes.
- Los distintos clientes no tienen que preocuparse de llevar a cabo ninguna tarea de mantenimiento, excepto la de establecer la misma CLASSPATH inicial.

Este tipo de instalación también tiene el inconveniente de aumentar el tiempo que tarda un cliente en iniciar una aplicación de Toolbox para Java. Asimismo, debe permitir que la CLASSPATH del cliente apunte al servidor. Puede emplear iSeries NetServer, que está integrado en OS/400, u otro método que le permita acceder a los archivos de iSeries NetServer, como iSeries Access para Windows.

Instalación de IBM Toolbox para Java

El método que utilice para instalar IBM Toolbox para Java dependerá de cómo desee gestionar la instalación. Una vez que haya decidido cómo desea gestionar la instalación, asegúrese de que el entorno cumpla los requisitos siguientes:

- Requisitos de OS/400
- Requisitos de estación de trabajo

Instalación de Toolbox para Java

Una vez que haya decidido cómo desea gestionar la instalación y que se haya asegurado de que se cumplan los requisitos para ejecutar IBM Toolbox para Java, puede instalar Toolbox para Java:

- Instalación de Toolbox para Java en el servidor
- Instalación de Toolbox para Java en la estación de trabajo

Requisitos de OS/400 para IBM Toolbox para Java

Una vez que haya decidido cómo desea gestionar la instalación, asegúrese de que el entorno OS/400 cumpla los requisitos siguientes:

- Opciones de OS/400 necesarias
- Dependencias de otros programas bajo licencia
- Compatibilidad con niveles distintos de OS/400
- Optimizaciones nativas al llevar a cabo la ejecución en la máquina virtual Java de OS/400
- »Requisitos para ejecutar aplicaciones de ToolboxME para iSeries«

Nota: antes de utilizar Toolbox para Java, compruebe que se cumplen los requisitos de estación de trabajo que corresponden a su entorno.

Opciones de OS/400 necesarias

Para ejecutar IBM Toolbox para Java en un entorno de cliente/servidor debe habilitar el perfil de usuario QUSER, iniciar los servidores de sistema principal y hacer que TCP/IP esté en ejecución:

- El perfil de usuario QUSER debe estar habilitado para iniciar los servidores de sistema principal.
- Los servidores de sistema principal escuchan y aceptan las peticiones de conexión de los clientes. La opción Servidores de sistema principal OS/400 (producto bajo licencia 5722SS1) se incluye con la opción base de OS/400. Para obtener más información, consulte el tema acerca de la administración de servidores de sistema principal.
- El soporte TCP/IP, que está integrado en OS/400, permite conectar el servidor a una red. Para obtener más información, consulte TCP/IP.

Cómo iniciar las opciones de OS/400 necesarias

En una línea de mandatos de iSeries, inicie las opciones de OS/400 necesarias siguiendo estos pasos:

1. Compruebe que el perfil QUSER está habilitado.
2. Para iniciar los servidores de sistema principal OS/400, utilice el mandato CL Iniciar servidor de sistema principal. Escriba **STRHOSTSVR *ALL** y pulse **INTRO**.
3. Para iniciar el servidor de gestión de datos distribuidos (DDM) TCP/IP, utilice el mandato CL Iniciar servidor TCP/IP. Escriba **STRTCPSVR SERVER(*DDM)** y pulse **INTRO**.

Cómo determinar si IBM Toolbox para Java está instalado en el servidor

Muchos servidores iSeries se distribuyen con el producto bajo licencia IBM Toolbox para Java ya instalado.

Para ver si Toolbox para Java ya está instalado, siga estos pasos:

- En iSeries Navigator, seleccione el sistema que desea emplear e inicie la sesión en él.
- En el **árbol de funciones** (el panel izquierdo), expanda el sistema y, a continuación, expanda **Configuración y servicio**.
- Expanda **Software** y, a continuación, expanda **Productos instalados**.
- En el panel **Detalles** (el panel derecho), examine la columna **Producto** para ver si en ella aparece 5722jc1. Si ve este producto, el programa bajo licencia IBM Toolbox para Java está instalado en el servidor seleccionado.

  Nota: también puede averiguar si Toolbox para Java está instalado ejecutando el mandato CL Ir a menú (**GO MENU(LICPGM)**), Opción 11.

Si Toolbox para Java no está instalado, puede instalar el producto bajo licencia IBM Toolbox para Java.

Si hay instalada una versión anterior de Toolbox para Java, primero suprima la versión que está instalada en este momento y, a continuación, instale el producto bajo licencia IBM Toolbox para Java. Para evitar posibles problemas, puede hacer una copia de seguridad de la versión de Toolbox para Java que está instalada en este momento antes de suprimirla.

Comprobar el perfil QUSER

Los servidores de sistema principal OS/400 se inician con el perfil de usuario QUSER, por lo que primero debe asegurarse de que el perfil QUSER está habilitado.

Comprobar el perfil QUSER

Para utilizar la línea de mandatos a fin de comprobar el perfil QUSER, siga estos pasos:

1. En una línea de mandatos de iSeries, escriba `DSPUSRPRF USRPRF(QUSER)` y pulse **Intro**.
2. Compruebe que **Estado** tiene establecido el valor *ENABLED. Si el estado del perfil no es *ENABLED, cambie el perfil QUSER.

Cambiar el perfil de usuario QUSER

Si el perfil QUSER no está establecido en *ENABLED, debe habilitarlo para iniciar los servidores de sistema principal OS/400. Asimismo, la contraseña del perfil QUSER no puede ser *NONE. Si ese es el caso, debe restablecerla.

Para utilizar la línea de mandatos a fin de habilitar el perfil QUSER, siga estos pasos:

1. Escriba `CHGUSRPRF USRPRF(QUSER)`.
2. Cambie el campo **Estado** a *ENABLED y pulse **INTRO**.

El perfil de usuario QUSER ahora está preparado para iniciar los servidores de sistema principal OS/400.

Dependencias de otros programas bajo licencia

En función de cómo desee utilizar IBM Toolbox para Java, puede que tenga que instalar otros programas bajo licencia. La información siguiente describe estas dependencias.

Visor de archivos en spool

Si desea utilizar las funciones del visor de archivos en spool (clase SpooledFileViewer) de IBM Toolbox para Java, asegúrese de que en el servidor esté instalada la opción 8 de sistema principal (Fonts de compatibilidad AFP).

Nota: las clases SpooledFileViewer, PrintObjectPageInputStream y PrintObjectTransformedInputStream solo funcionan cuando se conectan a sistemas cuya versión sea igual o posterior a V4R4.

SSL (capa de sockets segura)

Si desea utilizar SSL (capa de sockets segura), compruebe que tenga instalado lo siguiente:

- Programa bajo licencia IBM HTTP Server para iSeries, 5722-DG1
- OS/400 Opción 34 (Gestor de certificados digitales)
- IBM Cryptographic Access Provider de 128 bits para iSeries, 5722-AC3
- iSeries Client Encryption (128 bits), 5722-CE3

La versión V5R2 de IBM Toolbox para Java requiere que se utilice »la versión V5R1 o V5R2 de iSeries Client Encryption.«

Encontrará más información sobre SSL en el tema acerca de SSL (capa de sockets segura) y Java Secure Socket Extension.

Servidor HTTP para utilizar applets, servlets, SSL o AS400ToolboxInstaller

Si desea utilizar applets, servlets, SSL o la clase AS400ToolboxInstaller en el sistema iSeries, debe configurar un servidor HTTP e instalar los archivos de clase en el sistema iSeries. Encontrará más información sobre IBM HTTP Server en el manual IBM HTTP Server para AS/400 Guía del Webmaster, GC10-3129 (GC41-5434), en el siguiente URL: http://www.ibm.com/eserver/iseries/products/http/docs/doc.htm  . El manual Guía del Webmaster está disponible en formato HTML y PDF.

Si desea información sobre el gestor de certificados digitales y sobre cómo crear certificados digitales con IBM HTTP Server y trabajar con ellos, consulte Gestión de certificados digitales.

Compatibilidad con niveles distintos de OS/400

Dado que puede utilizar IBM Toolbox para Java tanto en el servidor como en el cliente, las cuestiones de compatibilidad afectan tanto a la ejecución en un servidor como a la conexión desde un cliente a un servidor.

Ejecutar IBM Toolbox para Java, Versión 5 Release 2, en servidores

Para instalar IBM Toolbox para Java (programa bajo licencia 5722-JC1 V5R2M0) en un sistema iSeries, el servidor debe ejecutar uno de los sistemas operativos siguientes:

- OS/400 Versión 5 Release 2
- OS/400 Versión 5 Release 1
- OS/400 Versión 4 Release 5

Puede instalar únicamente una versión del programa bajo licencia IBM Toolbox para Java en el sistema. Para instalar otra versión, primero elimine el programa bajo licencia IBM Toolbox para Java existente.

Utilizar IBM Toolbox para Java para conectarse desde un cliente a un servidor

Puede utilizar distintas versiones de IBM Toolbox para Java en un cliente y en el servidor al que se conecta. Para utilizar IBM Toolbox para Java Versión 5 Release 2 con el fin de acceder a los datos y recursos de un sistema iSeries, el **servidor al que se conecta** debe ejecutar uno de los sistemas operativos siguientes:

- OS/400 Versión 5 Release 2
- OS/400 Versión 5 Release 1
- OS/400 Versión 4 Release 5

La tabla siguiente muestra los requisitos de compatibilidad para la instalación de IBM Toolbox para Java y la conexión con distintas versiones anteriores de OS/400.

Modificación de Toolbox	Se distribuye con OS/400	LPP	Se instala en OS/400	Se conecta con versiones anteriores de OS/400
Mod 0	V4R2	5763-JC1 V3R2M0	V3R2 y superior	V3R2 y superior
Mod 1	V4R3	5763-JC1 V3R2M1	V3R2 y superior	V3R2 y superior
Mod 2	V4R4	5769-JC1 V4R2M0	V4R2 y superior	V4R2 y superior
Mod 3	V4R5	5769-JC1 V4R5M0	V4R3 y superior	V4R2 y superior
Mod 4	V5R1	5722-JC1 V5R1M0	V4R4 y superior	V4R3 y superior
Mod 5	V5R2	5722-JC1 V5R2M0	V4R5 y superior	V4R5 y superior

Optimizaciones nativas al llevar a cabo la ejecución en la máquina virtual Java de iSeries

Las optimizaciones nativas son un conjunto de funciones que hacen que las clases de IBM Toolbox para Java funcionen como un usuario esperaría que funcionaran al ejecutarse en OS/400. Las optimizaciones sólo inciden en el funcionamiento de IBM Toolbox para Java al ejecutarse en la máquina virtual Java de iSeries.

Es muy importante entender que los programas Java utilizan las optimizaciones nativas únicamente cuando se utiliza la versión de IBM Toolbox para Java que coincide con la versión de OS/400 en el servidor. Las optimizaciones son las siguientes:

- Inicio de sesión: si no se especifica ningún ID de usuario o ninguna contraseña en el objeto AS400, se utilizan el ID de usuario y la contraseña del trabajo actual.
- Llamada directa a las API de OS/400 en lugar de efectuar llamadas por socket a los servidores de sistema principal:
  - Acceso a base de datos a nivel de registro, colas de datos y espacio de usuario cuando se cumplen los requisitos de seguridad.
  - Llamada a programa y llamada a mandato cuando se cumplen los requisitos de seguridad y los requisitos de seguridad de hebras.

»**Nota:** para obtener un mejor rendimiento, establezca la propiedad driver de JDBC para utilizar el controlador nativo cuando el programa Java y el archivo de base de datos estén en el mismo sistema iSeries.«

No es necesario efectuar ningún cambio en la aplicación Java para obtener las optimizaciones. IBM Toolbox para Java habilita automáticamente las optimizaciones cuando corresponde.

Requisitos de compatibilidad de optimizaciones nativas

La tabla siguiente muestra qué versiones de IBM Toolbox para Java y OS/400 debe ejecutar para utilizar las optimizaciones nativas. Esta tabla trata únicamente las cuestiones de compatibilidad que afectan a las optimizaciones nativas. Para consultar aspectos generales de compatibilidad, vea Compatibilidad con niveles distintos de OS/400.

Nivel de OS/400	Modificación de la Caja de Herramientas necesaria para utilizar optimizaciones nativas				
V4R2	No hay mejoras de Mod0 disponibles.				
V4R3	Mod1	Mod2			
V4R4	Mod1	Mod2			
V4R5			Mod3		
V5R1				Mod4	
»V5R2					Mod5«

Para obtener las mejoras en el rendimiento, debe asegurarse de utilizar el archivo jar que contiene las optimizaciones nativas de OS/400. Para obtener más información, consulte la nota 1 de los archivos jar.

Si las versiones de IBM Toolbox para Java y OS/400 no coinciden, las optimizaciones nativas no están disponibles. En este caso, IBM Toolbox para Java funciona como si se ejecutara en un cliente.

Requisitos de ToolboxME para iSeries

La estación de trabajo, el dispositivo inalámbrico y el servidor deben cumplir determinados requisitos (indicados a continuación) para desarrollar y ejecutar aplicaciones de ToolboxME para iSeries. Aunque IBM Toolbox para Java 2 Micro Edition se considera parte de IBM Toolbox para Java, no se incluye en el producto bajo licencia.

ToolboxME para iSeries (jt400Micro.jar) se incluye en la versión de fuente abierto de Toolbox para Java, denominada JTOpen. Debe bajar y configurar ToolboxME para iSeries, que se encuentra en JTOpen, por separado.

Requisitos

Para utilizar ToolboxME para iSeries, la estación de trabajo, el dispositivo inalámbrico Tier0 y el servidor deben cumplir los requisitos siguientes.

Requisitos de estación de trabajo

Requisitos de estación de trabajo para desarrollar aplicaciones de ToolboxME para iSeries:

- Plataforma Java 2, Standard Edition, versión 1.3 o superior
- Máquina virtual Java para dispositivos inalámbricos
- Simulador o emulador de dispositivos inalámbricos

Requisitos de dispositivo inalámbrico

El único requisito para ejecutar aplicaciones de ToolboxME para iSeries en el dispositivo Tier0 consiste en utilizar una máquina virtual Java para dispositivos inalámbricos.

Requisitos de servidor

Requisitos de servidor para utilizar aplicaciones de ToolboxME para iSeries:

- MEServer, que se incluye en IBM Toolbox para Java o la última versión de JTOpen
- Requisitos de OS/400 para IBM Toolbox para Java

Requisitos de estación de trabajo para IBM Toolbox para Java

Una vez que haya decidido cómo desea gestionar la instalación, asegúrese de que la estación de trabajo cumpla los requisitos siguientes:

- Requisitos para ejecutar aplicaciones Java
- Requisitos para ejecutar applets Java
- Requisitos para desarrollar aplicaciones de ToolboxME para iSeries
- Requisitos de Swing

Nota: antes de utilizar Toolbox para Java, compruebe que se cumplen los requisitos de OS/400 que corresponden a su entorno.

Requisitos de estación de trabajo para ejecutar aplicaciones de Toolbox para Java

Para desarrollar y ejecutar aplicaciones de Toolbox para Java, asegúrese de que la estación de trabajo cumpla los requisitos siguientes:

- Recomendamos utilizar una máquina virtual Java (JVM) con soporte para la plataforma Java 2, Standard Edition (J2SE(TM)) o Enterprise Edition (J2EE(TM)), versiones 1.3.x o posteriores. Son muchas las nuevas funciones de Toolbox para Java que requieren utilizar esta versión de la máquina virtual Java. Sin embargo, todavía puede emplear una máquina virtual Java que ofrezca pleno soporte para JDK 1.1.8 o cualquier versión posterior de JDK, con la plataforma Java 2 incluida.
- Si el programa utiliza la Caja de Herramientas Gráfica o las clases del paquete vaccess, se requiere asimismo Swing 1.1. Los entornos que se han comprobado son los siguientes:
  - »Windows(R) 2000«
  - »Windows(R) XP«
  - AIX Versión 4.3.3.1
  - Sun Solaris(TM) Versión 5.7
  - »OS/400 Versión 4 Release 5 o posterior«
  - »Linux (Red Hat 7.0)«
- TCP/IP instalado y configurado.

Requisitos de estación de trabajo para ejecutar applets de IBM Toolbox para Java

Para desarrollar y ejecutar aplicaciones de Toolbox para Java, asegúrese de que la estación de trabajo cumpla los requisitos siguientes:

- Un navegador que tenga una máquina virtual Java (JVM) compatible. Los entornos que se han comprobado son los siguientes:
  - »Netscape Communicator 4.7, utilizando el conector de Java 1.3 o posterior

  Nota: IBM Toolbox para Java ya no soporta la ejecución en la máquina virtual Java por omisión en Netscape Navigator o Microsoft Internet Explorer. Para que el applet que utiliza las clases de Toolbox para Java se ejecute en un navegador, debe instalar un conector como el conector de Sun Java 2 Runtime Environment (JRE) 1.3.0  .«
- TCP/IP instalado y configurado.
- »La estación de trabajo debe conectarse a un servidor que ejecute OS/400 V4R5 o posterior.«

Requisitos de estación de trabajo para Swing para IBM Toolbox para Java

IBM Toolbox para Java pasó a dar soporte a Swing 1.1 en la versión V4R5 y este release sigue proporcionando ese soporte. Esta modificación requirió cambios de programación en las clases de IBM Toolbox para Java. Por consiguiente, si sus programas utilizan la Caja de Herramientas Gráfica o las clases del paquete vaccess de releases anteriores a V4R5, también tendrá que modificar sus programas.

Además de hacer cambios en la programación, las clases de Swing deben estar en la CLASSPATH al ejecutarse el programa. Las clases de Swing forman parte de la plataforma Java 2. Si no dispone de la plataforma Java 2, puede bajar las clases de Swing 1.1 de Sun Microsystems, Inc. 

Instalación de IBM Toolbox para Java en un servidor iSeries

Únicamente debe instalar IBM Toolbox para Java en el servidor iSeries si ha configurado como cliente el servidor o una partición del servidor.

Nota: la versión nativa de Toolbox para Java se distribuye con OS/400. Por consiguiente, si desea utilizar Toolbox para Java únicamente en el servidor iSeries, no es necesario que instale el producto bajo licencia. Para obtener más información sobre la versión nativa de Toolbox para Java, consulte la nota 1 de los archivos jar.

Antes de instalar IBM Toolbox para Java, debe asegurarse de que la versión de OS/400 cumple los requisitos para ejecutar Toolbox para Java. Asimismo, algunos servidores vienen preconfigurados con una instalación de Toolbox para Java. Puede resultarle de interés determinar si el producto bajo licencia Toolbox para Java ya está instalado en el servidor.

Instalación de Toolbox para Java

Puede instalar el programa bajo licencia IBM Toolbox para Java utilizando iSeries Navigator o la línea de mandatos.

Utilizar iSeries Navigator para instalar Toolbox para Java

Para instalar Toolbox para Java utilizando iSeries Navigator, siga estos pasos:

1. En iSeries Navigator, inicie la sesión en el sistema que desea emplear.
2. En el árbol de funciones (el panel izquierdo), expanda **Mis conexiones**.
3. Bajo **Mis conexiones**, pulse con el botón derecho el sistema donde desea instalar Toolbox para Java.
4. Seleccione **Ejecutar mandato**.
5. En el diálogo **Restaurar programa bajo licencia (RSTLICPGM)**, escriba la siguiente información y, a continuación, pulse **Aceptar**:
    - Producto: 5722JC1
    - Dispositivo: el nombre del dispositivo o archivo de salvar

    Nota: para obtener más información, pulse **Ayuda** en el diálogo **Restaurar programa bajo licencia (RSTLICPGM)**.

Puede emplear iSeries Navigator para ver el estado de la tarea Mandato de Management Central resultante siguiendo estos pasos:

1. Expanda **Management Central**.
2. Expanda **Actividad de tareas**.
3. Bajo **Actividad de tareas**, seleccione **Mandatos**.
4. En el panel de detalles, pulse la tarea de **Ejecutar mandato** adecuada.

Utilizar la línea de mandatos para instalar Toolbox para Java

Para instalar Toolbox para Java desde una línea de mandatos de iSeries, siga estos pasos:

1. En una línea de mandatos de iSeries, utilice el mandato CL Ir a menú. Escriba **GO MENU(LICPGM)** y pulse **INTRO**.
2. Seleccione **11.Instalar programa bajo licencia**.
3. Seleccione **5722-JC1 IBM Toolbox para Java**.

Encontrará más información sobre cómo instalar programas bajo licencia en Gestión de software y programas bajo licencia.

Instalación de IBM Toolbox para Java en la estación de trabajo

Antes de instalar Toolbox para Java, compruebe que se cumplen los requisitos de estación de trabajo que corresponden a su entorno. El método que utilice para instalar IBM Toolbox para Java en la estación de trabajo dependerá de cómo desee gestionar la instalación:

- Para instalar Toolbox para Java en clientes individuales, copie los archivos JAR en la estación de trabajo y configure la CLASSPATH de la estación de trabajo.
- Para utilizar el producto Toolbox para Java instalado en un servidor, sólo tiene que configurar la CLASSPATH de la estación de trabajo de modo que apunte a la instalación del servidor. Para que la CLASSPATH de la estación de trabajo apunte al servidor, el servidor debe tener instalado iSeries Netserver.

En esta información se describe cómo copiar los archivos de clase en la estación de trabajo. Para obtener más información sobre cómo establecer la CLASSPATH de la estación de trabajo, consulte la documentación del sistema operativo de la estación de trabajo o la información disponible en el sitio Web de Java de Sun.

Nota: para utilizar las clases de Toolbox para Java en la aplicación también es preciso que el sistema cumpla los requisitos de OS/400.

Los archivos de clase de Toolbox para Java están empaquetados en varios archivos jar, por lo que debe copiar uno o más de estos archivos jar en la estación de trabajo. Para obtener más información sobre qué archivos jar son necesarios para las funciones de Toolbox para Java específicas, consulte Archivos jar.

Ejemplo: cómo copiar jt400.jar

En el ejemplo que figura a continuación se supone que se desea copiar jt400.jar, que contiene las clases de núcleo de IBM Toolbox para Java.

Para copiar manualmente el archivo jar, siga estos pasos:

1. Busque el archivo jt400.jar en el directorio siguiente:
   /QIBM/ProdData/HTTP/Public/jt400/lib
2. Copie jt400.jar del servidor en la estación de trabajo. Para ello existen varios métodos:
   - Utilice iSeries Access para Windows a fin de correlacionar una unidad de red de la estación de trabajo con el servidor y, a continuación, copie el archivo.
   - Utilice el protocolo de transferencia de archivos (FTP) para enviar el archivo a la estación de trabajo (en modalidad binaria).
3. Actualice la variable de entorno CLASSPATH de la estación de trabajo.
   - Por ejemplo, si utiliza Windows NT y ha copiado jt400.jar en C:\jt400\lib, añada la serie siguiente al final de la variable CLASSPATH:

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

También tiene la opción de utilizar la versión de fuente abierto de Toolbox para Java, denominada JTOpen. Para obtener más información sobre JTOpen, consulte el sitio Web de IBM Toolbox para Java y JTOpen.

Si su navegador no está habilitado para Javascript, pulse el enlace de texto que figura junto a la imagen de Notas para enlazar con la nota adecuada; todas ellas se muestran a continuación de la tabla.

Archivos jar

IBM Toolbox para Java se distribuye con un conjunto de archivos jar. Cada uno de los archivos jar contiene paquetes Java que proporcionan funciones específicas. Puede reducir la cantidad de espacio de almacenamiento necesario utilizando únicamente los archivos jar que sean precisos para habilitar las funciones concretas que desee.

Para utilizar un archivo jar, compruebe que especifica una entrada para el mismo en la variable CLASSPATH.

El diagrama siguiente indica los archivos jar que debe haber en la variable CLASSPATH para utilizar las clases del paquete especificado.

Algunas entradas de la tabla tienen notas que facilitan más información. Si el navegador que utiliza está habilitado para Javascript, pulse la imagen para ver la información en una ventana independiente. De lo contrario, puede pulsar el enlace de texto para acceder a la misma información que se muestra a continuación de la tabla.

Paquete o función de Toolbox para Java	Archivos jar que deben estar en la variable CLASSPATH
Clases de acceso	jt400.jar (cliente) o jt400Native.jar (servidor) en el entorno de cliente/servidor habitual Nota 1, o jt400Proxy.jar en un entorno de proxy
CommandPrompter Nota 2	jt400.jar, jui400.jar, util400.jar Nota 3 y x4j400.jar
Clases HTML	jt400.jar Nota 1 más jt400Servlet.jar (cliente) o jt400Native.jar (servidor) Nota 1
GUI de origen de datos JDBC Nota 4	jt400.jar (cliente) Nota 1 y jui400.jar
Mensajes de error y del sistema de NLS	jt400Mri_lang_cntry.jar Nota 5
PCML (desarrollo) Nota 6	jt400.jar (cliente) o jt400Native.jar (servidor) Nota 1, Nota 7 y x4j400.jar
PCML (ejecución, serializado)	jt400.jar (cliente) o jt400Native.jar (servidor) Nota 1, Nota 7
PDML (desarrollo) Nota 2	uitools.jar, jui400.jar, util400.jar Nota 3 y x4j400.jar
PDML (ejecución, analizado) Nota 2	jui400.jar, util400.jar Nota 3 y x4j400.jar
PDML (ejecución, serializado) Nota 2	jui400.jar y util400.jar Nota 3
Clases ReportWriter	jt400.jar (cliente) o jt400Native.jar (servidor) Nota 1 y archivos jar de reportwriter Nota 8
Clases de recursos	jt400.jar (cliente) o jt400Native.jar (servidor) Nota 1
RFML	jt400.jar (cliente) o jt400Native.jar (servidor) Nota 1 y x4j400.jar
Clases de seguridad	jt400.jar (cliente) o jt400Native.jar (servidor) en el entorno de cliente/servidor habitual Nota 1, o jt400Proxy.jar en un entorno de proxy
Clases de servlets	jt400.jar Nota 1 más jt400Servlet.jar (cliente) o jt400Native.jar (servidor) Nota 1

Depurador del sistema iSeries Nota 2	jt400.jar (cliente) Nota 1 y tes.jar
ToolboxME para iSeries	jt400Micro.jar (cliente) y jt400.jar (servidor) o jt400Native.jar (servidor) Nota 9
Clases de vaccess	jt400.jar (cliente) Nota 1

Nota 1: algunas de las clases de IBM Toolbox para Java se encuentran en más de un archivo jar:

- **jt400.jar** - Clases de acceso, de recursos, de vaccess, de seguridad, PCML, RFML, soporte para JDBC y MEServer.
- **jt400.zip** - Utilice jt400.jar en lugar de jt400.zip. jt400.zip se distribuye para mantener la compatibilidad con los releases anteriores de IBM Toolbox para Java.
- **jt400Access.zip** - Clases de acceso, de recursos, de seguridad y PCML (es decir, las mismas clases que contiene jt400.jar menos las clases de vaccess). jtAccess400.zip se distribuye para mantener la compatibilidad con los releases anteriores de IBM Toolbox para Java. Utilice jt400.jar o jt400Native.jar en lugar de jt400Access.zip.
- **jt400Native.jar** - Accesos, recursos, seguridad, PCML, HTML, RFML, MEServer y optimizaciones nativas. Las optimizaciones nativas son un conjunto de clases (menos de 20) que aprovechan la función del iSeries cuando se ejecuta en la JVM del iSeries. Como jt400Native.jar contiene las optimizaciones nativas, al ejecutarse en la JVM del iSeries, utilice jt400Native.jar en lugar de jt400.jar. jt400Native.jar se distribuye con el OS/400 y reside en el directorio /QIBM/ProdData/OS400/jt400/lib.
- **jt400Native11x.jar** - Optimizaciones nativas únicamente. Si lleva a cabo la ejecución en la JVM del iSeries y de4sea utilizar jt400.jar, incluya jt400Native11x.jar en la variable CLASSPATH en lugar de jt400Native.jar. jt400Native11x.jar se distribuye con el OS/400 y reside en el directorio /QIBM/ProdData/OS400/jt400/lib.

Nota 2: para utilizar CommandPrompter, PDML o el depurador del sistema iSeries también se necesita un archivo jar adicional que no forma parte de Toolbox para Java: jhall.jar. Para obtener más información sobre cómo bajar jhall.jar, consulte el sitio Web de Sun JavaHelp(TM).

Nota 3: util400.jar contiene clases específicas del iSeries para dar formato a la entrada y para emplear el programa de solicitud de línea de mandatos (CL). Para utilizar la clase CommandPrompter se necesita util400.jar. Para utilizar PDML no se requiere util400.jar, pero es útil.

Nota 4: jui400.jar contiene las clases necesarias para utilizar la interfaz GUI DataSource JDBC. jt400.jar (Nota 1) contiene las clases necesarias para todas las demás funciones JDBC.

Nota 5: jt400Mri_xx_yy.jar contiene mensajes traducidos, entre los que se encuentran series incluidas en mensajes de excepción, diálogos y salida de otros procesos normales. En jt400Mri_lang_cntry.jar, lang = el código de idioma ISO y cntry = el código de país o región ISO utilizado para traducir el texto incluido. En algunos casos no se emplea el código de país o región ISO. Al instalar una versión de idioma concreta del programa bajo licencia IBM Toolbox para Java en el iSeries se instala el archivo jt400Mri_lang_cntry.jar adecuado. Si el idioma no está soportado, la instalación utiliza por omisión la versión en inglés, que se incluye en los archivos jar de IBM Toolbox para Java.

- Por ejemplo, al instalar la versión del idioma alemán del programa bajo licencia 5722-JC1, se instala el archivo jar del idioma alemán, jt400Mri_de.jar.

Puede añadir soporte para otros idiomas añadiendo más de uno de estos archivos jar a la variable classpath. Java carga la serie correcta según el entorno local actual.

Nota 6: la serialización del archivo PCML durante el desarrollo supone dos ventajas:

1. Tan solo deberá analizar el archivo PCML durante el desarrollo y no durante la ejecución.
2. Los usuarios necesitarán incluir menos archivos jar en la variable CLASSPATH para ejecutar la aplicación.

Para analizar el archivo PCML durante el desarrollo, necesitará el módulo de ejecución de PCML de data.jar o jt400.jar y el analizador PCML de x4j400.jar. Para ejecutar la aplicación serializada, los usuarios sólo necesitarán jt400.jar. Para obtener más información, consulte Construir llamadas a programa de iSeries con PCML.

Nota 7: utilice jt400.jar y jt400Native.jar en lugar de data400.jar. data400.jar contiene las clases de ejecución PCML, que ahora también están en jt400.jar y jt400Native.jar (Nota 1). data400.jar se distribuye para mantener la compatibilidad con los releases anteriores de IBM Toolbox para Java.

Nota 8: hay copias de las clases ReportWriter en más de un archivo jar:

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

Si su aplicación envía corrientes de datos PCL a un archivo en spool del iSeries, debe establecer como disponibles las clases de acceso mediante el archivo jar adecuado (Nota 1). Para crear un archivo en spool que contenga los datos PCL se necesitan las clases AS400, OutputQueue, PrintParameterList y SpooledFileOutputStream. Para obtener más información, consulte las clases ReportWriter.

Nota 9: jt400Micro.jar no contiene las clases necesarias para ejecutar MEServer, que residen en jt400.jar y jt400Native.jar (Nota 1). jt400Micro.jar sólo está disponible en el sitio Web de IBM Toolbox para Java y JTOpen.

Propiedades del sistema

Puede especificar propiedades del sistema para configurar diversos aspectos de IBM Toolbox para Java. Por ejemplo, puede utilizar las propiedades del sistema para definir un servidor proxy o un nivel de rastreo. Las propiedades del sistema son útiles para la adecuada configuración en tiempo de ejecución sin necesidad de volver a compilar el código. Las propiedades del sistema funcionan como las variables de entorno en el sentido de que cuando se cambia una propiedad del sistema durante la ejecución, por lo general el cambio no se refleja hasta la próxima vez que se ejecuta la aplicación.

Puede establecer las propiedades del sistema de varias formas:

- **Mediante el método java.lang.System.setProperties()**

  Puede establecer las propiedades del sistema de forma programática mediante el método java.lang.System.setProperties().

  Por ejemplo, el código siguiente establece la propiedad `com.ibm.as400.access.AS400.proxyServer` en `hqoffice`:

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

- **Mediante la opción -D del mandato java**

  Muchos entornos permiten establecer propiedades del sistema al ejecutar aplicaciones desde una línea de mandatos mediante la opción -D del mandato java.

  Por ejemplo, el programa siguiente ejecuta la aplicación denominada Inventory con la propiedad `com.ibm.as400.access.AS400.proxyServer` establecida en `hqoffice`:

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

- **Mediante un archivo jt400.properties**

  En algunos entornos, puede resultar poco práctico ordenar a todos los usuarios que establezcan sus propias propiedades del sistema. Como alternativa, puede especificar las propiedades del sistema de IBM Toolbox para Java en un archivo denominado jt400.properties donde se buscará como si fuera parte del paquete com.ibm.as400.access. Dicho de otro modo, coloque el archivo jt400.properties en un directorio com/ibm/as400/access al que apunte la sentencia CLASSPATH.

  Por ejemplo, establezca la propiedad `com.ibm.as400.access.AS400.proxyServer` en `hqoffice` insertando la línea siguiente en el archivo jt400.properties:

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

  El carácter de barra inclinada invertida (\) funciona como carácter de escape en los archivos de propiedades. Para especificar un carácter de barra inclinada invertida literal, emplee dos caracteres de barra inclinada invertida (\\).

  Modifique este ejemplo de un archivo jt400.properties para su entorno.

- **Mediante una clase Properties**

  Algunos navegadores no cargan los archivos de propiedades sin cambiar explícitamente los valores de seguridad. Sin embargo, la mayoría de los navegadores sí admiten propiedades en archivos .class, de modo que las propiedades del sistema de IBM Toolbox para Java también pueden especificarse mediante una clase denominada com.ibm.as400.access.Properties que amplía java.util.Properties.

  Por ejemplo, para establecer la propiedad `com.ibm.as400.access.AS400.proxyServer` en `hqoffice`, utilice el código Java siguiente:

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

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

Modifique y compile este ejemplo de un archivo fuente Properties.java para su entorno.

Si una propiedad del sistema de IBM Toolbox para Java se establece mediante más de uno de los procedimientos descritos anteriormente, se aplica la prioridad siguiente (por orden de prioridad decreciente):

1. La propiedad del sistema establecida programáticamente mediante `java.lang.System.setProperties()`
2. La propiedad del sistema establecida mediante la opción -D del mandato java
3. La propiedad del sistema establecida mediante una clase Properties
4. La propiedad del sistema establecida mediante un archivo jt400.properties

IBM Toolbox para Java da soporte a las propiedades del sistema siguientes:

- Propiedades de servidor proxy
- Propiedades de rastreo
- Propiedades de llamada a mandato/programa

Propiedades de servidor proxy

Propiedad de servidor proxy	Descripción
com.ibm.as400.access.AS400.proxyServer	Especifica el nombre de sistema principal y el número de puerto de servidor proxy, con el formato: `nombresisprincipal:númeropuerto` El número de puerto es opcional.
com.ibm.as400.access.SecureAS400.proxyEncryptionMode	Especifica qué parte del flujo de datos de proxy se cifra mediante SSL. Los valores válidos son: • 1 = De cliente proxy a servidor proxy • 2 = De servidor proxy a iSeries • 3 = De cliente proxy a servidor proxy y de servidor proxy a iSeries
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval	Especifica con qué frecuencia, en segundos, el servidor proxy busca si hay conexiones desocupadas. El servidor proxy inicia una hebra para buscar los clientes que ya no tienen comunicación. Utilice esta propiedad para establecer con qué frecuencia busca la hebra si hay conexiones desocupadas.

com.ibm.as400.access.TunnelProxyServer.clientLifetime	Especifica durante cuánto tiempo, en segundos, puede estar desocupado un cliente antes de que el servidor proxy elimine las referencias a los objetos de modo que la máquina virtual Java pueda eliminarlos en la recogida de basura. El servidor proxy inicia una hebra para buscar los clientes que ya no tienen comunicación. Utilice esta propiedad para establecer durante cuánto tiempo puede estar desocupado un cliente antes de que se ejecute la recogida de basura sobre él.

Propiedades de rastreo

Propiedad de rastreo	Descripción
com.ibm.as400.access.Trace.category	Especifica qué categorías de rastreo deben habilitarse. Se trata de una lista delimitada por comas que contiene cualquier combinación de categorías de rastreo. La lista completa de categorías de rastreo se define en la clase Trace.
com.ibm.as400.access.Trace.file	Especifica el archivo en que se escribe la salida de rastreo. Por omisión la salida de rastreo se escribe en System.out.
com.ibm.as400.access.ServerTrace.JDBC	Especifica qué categorías de rastreo deben iniciarse en el trabajo servidor JDBC. Para obtener información sobre los valores soportados, consulte la propiedad de rastreo del servidor JDBC.

Propiedades de llamada a mandato/programa

Propiedad de llamada a mandato/programa	Descripción
com.ibm.as400.access.CommandCall.threadSafe	Especifica si debe suponerse que las llamadas a mandato son seguras en ejecución multihebra. Si es `true`, se supone que todas las llamadas a mandato son seguras en ejecución multihebra. Si es `false`, se supone que todas las llamadas a mandato no son seguras en ejecución multihebra. Esta propiedad no se tiene en cuenta para un objeto CommandCall determinado si `CommandCall.setThreadSafe(true/false)` o `AS400.setMustUseSockets(true)` se ha llevado a cabo sobre el objeto.
com.ibm.as400.access.ProgramCall.threadSafe	Especifica si debe suponerse que las llamadas a programa son seguras en ejecución multihebra. Si es `true`, se supone que todas las llamadas a programa son seguras en ejecución multihebra. Si es `false`, se supone que todas las llamadas a programa no son seguras en ejecución multihebra. Esta propiedad no se tiene en cuenta para un objeto ProgramCall determinado si `ProgramCall.setThreadSafe(true/false)` o `AS400.setMustUseSockets(true)` se ha llevado a cabo sobre el objeto.

Ejemplos simples de programación

Estos ejemplos muestran algunas de los procedimientos por los que puede empezar a codificar sus propios programas Java utilizando las clases de IBM Toolbox para Java. Estos ejemplos, pensados para los programadores que simplemente comienzan a utilizar las clases de Toolbox para Java, contienen descripciones detalladas sobre las líneas clave del código.

Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente. (Para ver las notas en una ventana emergente, el navegador debe dar soporte a JavaScript.)
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Si desea obtener ayuda sobre cómo empezar, consulte cómo escribir el primer programa de Toolbox para Java.

Para obtener enlaces con muchos de los demás ejemplos proporcionados en la información de Toolbox para Java, consulte los ejemplos de código.

Utilice la lista siguiete para ver los ejemplos simples de programación:

- Llamadas a mandatos
- Cómo se utilizan las colas de mensajes
- Cómo se utilizan las clases de acceso a nivel de registro
- Cómo se utilizan las clases JDBC para crear y llenar con datos una tabla
- Visualizar una lista de trabajos servidores en una GUI

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Clases de IBM Toolbox para Java

Las clases de IBM Toolbox para Java se clasifican (como todas las clases Java) en paquetes. Cada paquete proporciona un tipo determinado de funcionalidad. Para mayor comodidad, en esta documentación habitualmente se hace referencia a cada paquete con un nombre corto. Por ejemplo, el paquete com.ibm.as400.access se denomina simplemente paquete access.

Utilice los enlaces de la lista siguiente para encontrar información sobre las clases de los distintos paquetes de Toolbox para Java:

- Las clases de acceso permiten acceder a los recursos del iSeries y gestionarlos.
- Las clases HTML permiten crear rápidamente formularios y tablas HTML.
- »Las clases micro permiten crear programas Java que dan a los dispositivos inalámbricos acceso directo a los datos y servicios del servidor iSeries.«
- Las clases ReportWriter permiten crear documentos con formato a partir de orígenes de datos XML.
- Las clases de recursos utilizan una infraestructura común para acceder a los recursos del iSeries y gestionarlos.
- Las clases de seguridad permiten establecer conexiones seguras con el servidor y verificar la identidad de los usuarios que trabajan en el sistema iSeries.
- Las clases de servlets ayudan a recuperar y formatear datos para utilizarlos en los servlets Java.
- Las clases de utilidades permiten realizar tareas administrativas; por ejemplo, utilizar la clase AS400ToolboxInstaller y la clase AS400JarMaker.
- Las clases de vaccess permiten presentar visualmente los datos y manipularlos.

Clases de acceso de IBM Toolbox para Java

Las clases de acceso de IBM Toolbox para Java representan datos y recursos de iSeries. Las clases funcionan con servidores iSeries y AS/400e para proporcionar una interfaz habilitada para Internet que permita acceder y actualizar los datos y recursos de servidor.

Las clases que proporcionan acceso a los recursos de iSeries y AS/400e son las siguientes:

- AS400 - gestiona información de inicio de sesión, crea y mantiene conexiones por socket, y envía y recibe datos
- SecureAS400 - permite utilizar un objeto AS400 al enviar o recibir datos cifrados
- AS400JPing - permite al programa Java consultar los servidores de sistema principal para ver qué servicios están en ejecución y qué puertos están en servicio
- BidiTransform - permite realizar las propias conversiones de texto bidireccional
- »Clases de tablas hash agrupadas en clúster - permite al programa Java compartir y duplicar datos no persistentes entre los nodos en tablas hash agrupadas en clúster de gran disponibilidad«
- Llamada a mandato - ejecuta mandatos iSeries de proceso por lotes
- Agrupación de conexiones - gestiona una agrupación de objetos AS400, que se utiliza para compartir conexiones y gestionar el número de conexiones que un usuario puede tener con un servidor iSeries
- Área de datos - crea, accede a y suprime áreas de datos
- Conversión y descripción de datos - convierte y maneja datos, y describe el formato de registro de un almacenamiento intermedio de los datos
- Colas de datos - crea, accede a, cambia y suprime colas de datos
- Certificados digitales - gestiona certificados digitales en servidores iSeries
- Variable de entorno - gestiona las variables de entorno de iSeries
- Anotaciones de eventos - permite anotar excepciones y mensajes con independencia del dispositivo utilizado para visualizarlos
- Excepciones - lanza errores cuando, por ejemplo, se producen errores de dispositivo o de programación
- FTP - proporciona una interfaz programable con las funciones FTP
- Sistema de archivos integrado - accede a archivos, abre archivos, abre corrientes de datos de entrada y salida, y lista el contenido de los directorios
- Llamada a aplicación Java - efectúa una llamada a un programa Java existente en un servidor iSeries que se ejecuta en la máquina virtual Java de iSeries
- JDBC - accede a datos de DB2 UDB para iSeries
- Trabajos - accede a trabajos y anotaciones de trabajo de iSeries
- Mensajes - accede a mensajes y colas de mensajes en el sistema iSeries
- Configuración de NetServer - accede al estado y la configuración de iSeries NetServer y modifica esta información
- Permiso - visualiza y cambia las autorizaciones para los objetos de un servidor iSeries
- Imprimir - manipula los recursos de impresión de iSeries
- Licencia de producto - gestiona las licencias de productos iSeries
- Llamada a programa - efectúa una llamada a un programa de iSeries
- Nombre de vía de acceso a objeto de QSYS - representa los objetos que hay en el sistema de archivos integrado de iSeries

- Acceso a nivel de registro - crea, lee, actualiza y suprime archivos y miembros de iSeries
- Llamada a programa de servicio - efectúa una llamada a un programa de servicio iSeries
- Estado del sistema - visualiza información de estado del sistema y permite acceder a la información de agrupación del sistema
- Valores del sistema - recupera y cambia valores del sistema y atributos de red
- Rastreo (facilidad de mantenimiento) - anota puntos de rastreo y mensajes de diagnóstico
- Usuarios y grupos - accede a los usuarios y grupos de iSeries
- Espacio de usuario - accede a un espacio de usuario de iSeries

Nota: Toolbox para Java proporciona un segundo conjunto de clases, denominadas clases de recursos, para trabajar con objetos y listas de iSeries. Las clases de recursos presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación.

Clase AS400

La clase AS400 gestiona estos elementos:

- Un conjunto de conexiones por socket establecidas con los trabajos servidores existentes en el servidor iSeries.
- El comportamiento de inicio de sesión correspondiente al servidor. Ello incluye la presentación al usuario de la solicitud de información de inicio de sesión, la colocación de contraseñas en antememoria y la gestión de usuario por omisión.

El programa Java debe proporcionar un objeto AS400 cuando utiliza una instancia de una clase que accede al iSeries. Por ejemplo, el objeto CommandCall requiere un objeto AS400 para poder enviar mandatos al iSeries.

El objeto AS400, cuando se ejecuta en la máquina virtual Java de iSeries, maneja de manera diferente las conexiones, los ID de usuario y las contraseñas. Encontrará más información en Máquina virtual Java de iSeries.

»Los objetos AS400 ahora soportan la autenticación de kerberos, utilizando la interfaz de programación de aplicaciones Java Generic Security Service (API JGSS) para autenticar en el servidor, en lugar de utilizar un ID de usuario y una contraseña.

Nota: para utilizar los tickets de kerberos debe instalarse J2SDK v1.4 y configurarse la interfaz de programación de aplicaciones Java Generic Security Services (JGSS). Para obtener más información sobre JGSS, consulte la documentación de seguridad de J2SDK v1.4 .«

En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones con el iSeries mediante el objeto AS400. En AS400ConnectionPool encontrará información acerca de cómo reducir el tiempo de conexión inicial solicitando conexiones desde una agrupación de conexiones.

La clase AS400 proporciona las siguientes funciones de inicio de sesión:

- Autenticar el perfil de usuario
- »Obtener una credencial de símbolo de perfil y autenticar el perfil de usuario asociado«
- »Establecer una credencial de símbolo de perfil«
- Gestionar identificadores de usuario por omisión
- Colocar contraseñas en antememoria
- Solicitar ID de usuario
- Cambiar una contraseña
- Obtener la versión y el release del iSeries

Para obtener información acerca de cómo utilizar un objeto AS400 al enviar o recibir datos cifrados, consulte la clase SecureAS400.

Clase SecureAS400

Cuando un objeto AS400 se comunica con el servidor, los datos de usuario (excepto la contraseña de usuario) se envían sin cifrar al servidor. Por consiguiente, los objetos de IBM Toolbox para Java asociados a un objeto AS400 intercambian datos con el servidor por medio de una conexión normal.

Si desea utilizar IBM Toolbox para Java para intercambiar datos delicados con el servidor, puede cifrar los datos mediante SSL (capa de sockets segura). Utilice el objeto SecureAS400 para designar los datos que desea cifrar. Los objetos de IBM Toolbox para Java asociados a un objeto SecureAS400 intercambian datos con el servidor por medio de una conexión segura.

» Para obtener más información, consulte el tema acerca de SSL (capa de sockets segura) y Java Secure Socket Extension. «

La clase SecureAS400 es una subclase de la clase AS400.

Puede configurar una conexión de servidor segura creando una instancia de un objeto SecureAS400 de los modos siguientes:

- SecureAS400(String systemName, String userID) le solicita información de inicio de sesión
- SecureAS400(String systemName, String userID, String password) no le solicita información de inicio de sesión

A continuación figura un ejemplo de cómo se utiliza CommandCall para enviar mandatos al sistema iSeries utilizando una conexión segura:

```
// Cree un objeto AS400 seguro. Esta sentencia es la única que cambia
// en relación con el caso no SSL.
SecureAS400 sys = new SecureAS400("mySystem.myCompany.com");

// Cree un objeto llamada a mandato.
CommandCall cmd = new CommandCall(sys, "myCommand");

// Ejecute los mandatos. Se establece una conexión segura al
// ejecutarse el mandato. Toda la información pasada entre el
// cliente y el servidor está cifrada.
cmd.run();
```

AS400JPing

La clase AS400JPing permite al programa Java consultar los servidores de sistema principal para ver qué servicios están en ejecución y qué puertos están en servicio. Para consultar los servidores desde una línea de mandatos, utilice la clase JPing.

La clase AS400JPing proporciona varios métodos:

- Realizar un ping del servidor
- Realizar un ping de un servicio específico en el servidor
- Establecer un objeto PrintWriter en el que desea anotar la información de la operación ping
- Establecer el tiempo de espera de la operación ping

Ejemplo: Utilización de AS400JPing dentro de un programa Java para realizar un ping del servicio de mandatos remotos de iSeries:

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

Clase BidiTransform

La clase AS400BidiTransform proporciona transformaciones de diseño que hacen posible la conversión de texto bidireccional en formato de iSeries (tras convertirlo primero a Unicode) a texto bidireccional en formato Java, o de formato Java a formato de iSeries.

La clase AS400BidiTransform permite llevar a cabo estas acciones:

- Obtener y establecer el CCSID de sistema
- Obtener y establecer el tipo de serie de los datos de iSeries
- Obtener y establecer el tipo de serie de los datos Java
- Convertir datos de un diseño Java a iSeries
- Convertir datos de un diseño de iSeries a Java

Ejemplo: cómo se utiliza la clase AS400BidiTransform para transformar texto bidireccional

El ejemplo que sigue muestra cómo se puede utilizar la clase AS400BidiTransform para transformar texto bidireccional:

```
// Datos Java a diseño de iSeries:
AS400BidiTransform abt;
abt = new AS400BidiTransform(424);
String dst = abt.toAS400Layout("alguna serie bidireccional");
```

Clases ClusteredHashTable

Las clases ClusteredHashTable permiten a los programas Java utilizar tablas hash agrupadas en clúster de gran disponibilidad para compartir y duplicar datos en almacenamiento no persistente entre los nodos de un clúster. Para utilizar las clases ClusteredHashTable, compruebe que puede emplear almacenamiento no persistente para los datos. Los datos duplicados no están cifrados.

Nota: la siguiente información da por supuesto que se han entendido los conceptos y términos habituales en la agrupación en clúster de iSeries. Para obtener más información sobre los clústeres y cómo emplearlos, consulte Clústeres.

Para utilizar la clase ClusteredHashTable es necesario haber definido y activado un clúster en los sistemas iSeries. También debe iniciar un servidor de tablas hash agrupadas en clúster. Para obtener más información, consulte Configurar clústeres e Interfaces API de tablas hash agrupadas en clúster.

Los parámetros obligatorios son el nombre del servidor de tablas hash agrupadas en clúster y el objeto AS400, que representa el sistema que contiene el servidor de tablas hash agrupadas en clúster.

Para almacenar datos en un servidor de tablas hash agrupadas en clúster, necesita un handle de conexión y una clave:

- Al abrir una conexión, el servidor de tablas hash agrupadas en clúster asigna el handle de conexión que debe especificar en las peticiones posteriores efectuadas al servidor de tablas hash agrupadas en clúster. Este handle de conexión sólo es adecuado para el objeto AS400 del que se ha creado una instancia, por lo que debe abrir otra conexión si utiliza un objeto AS400 distinto.
- Debe especificar la clave para acceder a los datos de la tabla hash agrupada en clúster y cambiar los datos. Las claves duplicadas no están soportadas.

La clase ClusteredHashTable proporciona métodos que permiten llevar a cabo las acciones siguientes:

- Abrir una conexión con el trabajo servidor de tablas hash agrupadas en clúster
- Generar una clave exclusiva para almacenar datos en la tabla hash agrupada en clúster
- Cerrar la conexión activa con el trabajo servidor de tablas hash agrupadas en clúster

Algunos métodos de la clase ClusteredHashTable utilizan la clase ClusteredHashTableEntry para llevar a cabo las acciones siguientes:

- Obtener una entrada de la tabla hash agrupada en clúster
- Almacenar una entrada en la tabla hash agrupada en clúster
- Obtener una lista de entradas de la tabla hash agrupada en clúster para todos los perfiles de usuario

El ejemplo siguiente se ejecuta en el servidor de tablas hash agrupadas en clúster denominado CHTSVR01. Supone que ya hay un clúster y un servidor de tablas hash agrupadas en clúster activos. Abre una conexión, genera una clave, coloca una entrada utilizando la clave nueva en la tabla hash agrupada en clúster, obtiene una entrada de la tabla hash agrupada en clúster y cierra la conexión.

```
     ClusteredHashTableEntry myEntry = null;

     String myData = new String("Estos son mis datos.");
     System.out.println("Datos que deben almacenarse: " + myData);

AS400 system = new AS400();

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

```
// Abra una conexión.
cht.open();

// Obtenga una clave para la tabla hash.
byte[] key = null;
key = cht.generateKey();

// Prepare algunos datos que desee almacenar en la tabla hash.
// ENTRY_AUTHORITY_ANY_USER significa que cualquier usuario puede
// acceder a la entrada de la tabla hash agrupada en clúster.
// DUPLICATE_KEY_FAIL significa que si la clave especificada ya existe,
// la petición ClusteredHashTable.put() no se ejecutará correctamente.
int timeToLive = 500;
myEntry = new ClusteredHashTableEntry(key,myData.getBytes(),timeToLive,
     ClusteredHashTableEntry.ENTRY_AUTHORITY_ANY_USER,
     ClusteredHashTableEntry.DUPLICATE_KEY_FAIL);

// Almacene (o coloque) la entrada en la tabla hash.
cht.put(myEntry);

// Obtenga una entrada de la tabla hash.
ClusteredHashTableEntry output = cht.get(key);

// Cierre la conexión.
cht.close();
```

La utilización de la clase ClusteredHashTable hace que el objeto AS400 se conecte al servidor. Para obtener más información, consulte Gestionar conexiones.«

Llamada a mandato

La clase CommandCall permite a un programa Java llamar a un mandato iSeries no interactivo. Los resultados del mandato están disponibles en una lista de objetos AS400Message.

La entrada de CommandCall es:

- La serie del mandato que ha de ejecutarse
- El objeto AS400 que representa el sistema que va a ejecutar el mandato

La serie del mandato se puede establecer en el constructor, mediante el método setCommand(), o se puede establecer en el método run(). Una vez ejecutado el mandato, el programa Java puede emplear el método getMessageList() para recuperar los mensajes de iSeries que se hayan producido como consecuencia del mandato.

La utilización de la clase CommandCall hace que el objeto AS400 se conecte al iSeries.

El ejemplo que sigue muestra cómo se utiliza la clase CommandCall para ejecutar un mandato en un sistema iSeries:

```
// Cree un objeto AS400.
 AS400 sys = new AS400("mySystem.myCompany.com");

     // Cree un objeto llamada a mandato. Este
     // programa establece que el mandato se
     // ejecute más adelante. Se le podría establecer
     // aquí en el constructor.
 CommandCall cmd = new CommandCall(sys);

     // Ejecute el mandato CRTLIB.
 cmd.run("CRTLIB MYLIB");

     // Obtenga la lista de mensajes
     // que contiene el resultado
     // del mandato.
 AS400Message[] messageList = cmd.getMessageList();

     // ...Procese la lista de mensajes.

     // Desconecte, puesto que ya ha terminado
     // de enviar mandatos al servidor.
 sys.disconnectService(AS400.COMMAND);
```

La utilización de la clase CommandCall hace que el objeto AS400 se conecte al iSeries. En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones.

»Cuando el programa Java y el mandato del servidor iSeries están en el mismo servidor, por omisión Toolbox para Java consulta la seguridad de la hebra para el mandato en el sistema.« Si es segura, el mandato se ejecuta en la hebra. Puede suprimir la consulta en tiempo de ejecución especificando explícitamente la seguridad de la hebra para el mandato con el método setThreadSafe().

Ejemplo

Ejecutar un mandato especificado por el usuario.

Agrupación de conexiones

Utilice las agrupaciones de conexiones para compartir conexiones y gestionar conjuntos (agrupaciones) de conexiones con un servidor iSeries. Por ejemplo, una aplicación puede recuperar una conexión de una agrupación, utilizarla y, a continuación, devolverla a la agrupación para volver a utilizarla.

La clase AS400ConnectionPool gestiona una agrupación de objetos AS400. La clase AS400JDBCConnectionPool representa una agrupación de AS400JDBCConnections disponibles para su uso por un programa Java como parte del soporte de la Caja de Herramientas para la API JDBC 2.0 Optional Package.» La interfaz JDBC ConnectionPool también está soportada en la API JDBC 3.0, que está empaquetada con la plataforma Java 2, Standard Edition, versión 1.4.«

Una agrupación de conexiones, de cualquiera de los dos tipos posibles, hace un seguimiento del número de conexiones que crea. Con los métodos heredados de ConnectionPool, puede establecer diversas propiedades de agrupación de conexiones, entre las que se incluyen las siguientes:

- el número máximo de conexiones que una agrupación puede proporcionar
- el tiempo máximo de vida de una conexión
- el tiempo máximo de inactividad de una conexión

En términos de rendimiento, la conexión con el servidor es una operación cara. El uso de agrupaciones de conexiones puede aumentar el rendimiento al evitar tiempos de conexión repetidos. Por ejemplo, cree conexiones cuando cree la agrupación de conexiones rellenando la agrupación con conexiones activas (preconectadas). En lugar de crear nuevas conexiones, puede utilizar una agrupación de conexiones para recuperar, utilizar, devolver y volver a utilizar fácilmente los objetos de conexión.

»Puede recuperar una conexión mediante una AS400ConnectionPool especificando el nombre del sistema, el ID de usuario, la contraseña y (si lo desea) el servicio.« Para especificar el servicio al que desea conectar, utilice las constantes de la clase AS400 (FILE, PRINT, COMMAND, etc.).

Tras recuperar y utilizar la conexión, las aplicaciones devuelven las conexiones a la agrupación. Es cada aplicación la responsable de devolver las conexiones a la agrupación para que vuelvan a emplearse. Si las conexiones no se devuelven a la agrupación, el tamaño de la agrupación de conexiones sigue creciendo y las conexiones no se reutilizan.

Consulte Gestión de conexiones para obtener más información acerca de cómo gestionar cuándo se abre una conexión con el iSeries al utilizar las clases AS400ConnectionPool.

Ejemplo: cómo se utiliza una AS400ConnectionPool para reutilizar objetos AS400

Área de datos

La clase DataArea es una clase base abstracta que representa un objeto área de datos de iSeries. Esta clase base tiene cuatro subclases que dan soporte a estos elementos: datos de tipo carácter, datos decimales, datos lógicos y áreas de datos locales que contienen datos de tipo carácter.

Mediante la clase DataArea se pueden realizar estas tareas:

- Obtener el tamaño del área de datos
- Obtener el nombre del área de datos
- Devolver el objeto de sistema AS400 correspondiente al área de datos
- Renovar los atributos del área de datos
- Establecer el sistema en el que existe el área de datos

La utilización de la clase DataArea hace que el objeto AS400 se conecte al servidor. En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones.

CharacterDataArea

La clase CharacterDataArea representa un área de datos existente en el servidor que contiene datos de tipo carácter. Las áreas de datos de tipo carácter no disponen de un servicio que permita identificar los datos con el debido CCSID; por lo tanto, el objeto área de datos presupone que el CCSID de los datos es el del usuario. En la escritura, el objeto área de datos se convierte de un tipo serie (Unicode) al CCSID del usuario antes de que los datos se escriban en el servidor. En la lectura, el objeto área de datos presupone que el CCSID de los datos es el del usuario y la conversión se realiza desde dicho CCSID a Unicode antes de devolver la serie al programa. Cuando se leen datos del área de datos, la cantidad de datos leídos viene expresada en el número de caracteres, no en el número de bytes.

Mediante la clase CharacterDataArea se pueden realizar estas tareas:

- Borrar el área de datos para que sólo contenga blancos
- Crear en el sistema un área de datos de tipo carácter utilizando los valores por omisión de las propiedades
- Crear un área de datos de tipo carácter con los atributos especificados
- Suprimir el área de datos del sistema en el que existe
- Devolver el nombre de vía de acceso del sistema de archivos integrado del objeto representado por el área de datos
- Leer la totalidad de los datos contenidos en el área de datos
- Leer una cantidad especificada de datos del área de datos, a partir del desplazamiento 0 o del desplazamiento que se haya especificado
- Establecer el nombre totalmente calificado de la vía de acceso del sistema de archivos integrado del área de datos
- Escribir datos al principio del área de datos
- Escribir una cantidad especificada de datos en el área de datos, a partir del desplazamiento 0 o del desplazamiento que se haya especificado

DecimalDataArea

La clase DecimalDataArea representa un área de datos existente en el servidor que contiene datos decimales.

Mediante la clase DecimalDataArea se pueden realizar estas tareas:

- Borrar el área de datos para que contenga 0,0
- Crear en el sistema un área de datos decimales utilizando los valores por omisión de las propiedades
- Crear un área de datos decimales con los atributos especificados
- Suprimir el área de datos del servidor en el que existe
- Devolver el número de dígitos que hay en el área de datos a la derecha de la coma decimal
- Devolver el nombre de vía de acceso del sistema de archivos integrado del objeto representado por el área de datos
- Leer la totalidad de los datos contenidos en el área de datos
- Establecer el nombre totalmente calificado de la vía de acceso del sistema de archivos integrado del área de datos
- Escribir datos al principio del área de datos

El siguiente ejemplo muestra cómo se crea un área de datos decimales y cómo se escribe en ella:

```
       // Establezca una conexión con el servidor "My400".
    AS400 system = new AS400("MyServer");
       // Cree un objeto DecimalDataArea.
    QSYSObjectPathName path = new QSYSObjectPathName("MYLIB", "MYDATA",
"DTAARA");
    DecimalDataArea dataArea = new DecimalDataArea(system, path.getPath());
       // Cree en el servidor el área de datos decimales utilizando los
valores por omisión.
    dataArea.create();
       // Borre el área de datos.
    dataArea.clear();
       // Escriba en el área de datos.
    dataArea.write(new BigDecimal("1,2"));
       // Lea el área de datos.
    BigDecimal data = dataArea.read();
       // Suprima el área de datos del servidor.
    dataArea.delete();
```

LocalDataArea

La clase LocalDataArea representa un área de datos local existente en el servidor. En el servidor existe un área de datos local como área de datos de tipo carácter, pero el área de datos local tiene algunas restricciones que deben tenerse en cuenta.

El área de datos local está asociada a un trabajo servidor y no es posible acceder a ella desde otro trabajo. Por lo tanto, esta área de datos local no se puede crear ni suprimir. Cuando el trabajo servidor finaliza, el área de datos local asociada a dicho trabajo se suprime automáticamente, y el objeto LocalDataArea que hace referencia al trabajo deja de ser válido. También es preciso tener presente que las áreas de datos locales tienen un tamaño fijo de 1024 caracteres en el servidor.

Mediante la clase LocalDataArea se pueden realizar estas tareas:

- Borrar el área de datos para que sólo contenga blancos
- Leer la totalidad de los datos contenidos en el área de datos
- Leer una cantidad especificada de datos del área de datos, a partir del desplazamiento que se haya especificado
- Escribir datos al principio del área de datos
- Escribir una cantidad especificada de datos en el área de datos, escribiéndose el primer carácter en el

desplazamiento

LogicalDataArea

La clase LogicalDataArea representa un área de datos existente en el servidor que contiene datos lógicos.

Mediante la clase LogicalDataArea puede realizar estas tareas:

- Borrar el área de datos para que contenga false
- Crear en el servidor un área de datos de tipo carácter utilizando los valores por omisión de las propiedades
- Crear un área de datos de tipo carácter con los atributos especificados
- Suprimir el área de datos del servidor en el que existe
- Devolver el nombre de vía de acceso del sistema de archivos integrado del objeto representado por el área de datos
- Leer la totalidad de los datos contenidos en el área de datos
- Establecer el nombre totalmente calificado de la vía de acceso del sistema de archivos integrado del área de datos
- Escribir datos al principio del área de datos

DataAreaEvent

La clase DataAreaEvent representa un evento de área de datos.

La clase DataAreaEvent se puede usar con cualquiera de las clases DataArea. Mediante la clase DataAreaEvent puede realizar esta tarea:

- Obtener el identificador del evento

DataAreaListener

La clase DataAreaListener proporciona una interfaz que permite recibir eventos de área de datos.

La clase DataAreaListener se puede usar con cualquiera de las clases DataArea. Es posible invocar la clase DataAreaListener cuando se lleva a cabo cualquiera de estas acciones:

- Borrar
- Crear
- Suprimir
- Leer
- Escribir

Conversión y descripción de datos

Las clases de **conversión de datos** proporcionan la posibilidad de convertir datos de tipo carácter y numérico entre los formatos de iSeries y Java. La conversión puede ser necesaria cuando se accede a datos de iSeries desde un programa Java. Las clases de conversión de datos soportan la conversión de diversos formatos numéricos y entre diversas páginas de códigos EBCDIC y Unicode.

Las clases de **descripción de datos** se construyen sobre las clases de conversión de datos para convertir todos los campos de un registro con una sola llamada de método. La clase RecordFormat permite al programa describir los datos que constituyen un objeto DataQueueEntry, un parámetro ProgramCall, un registro de un archivo de base de datos al que se accede mediante clases de acceso a nivel de registro o cualquier almacenamiento intermedio de datos de iSeries. La clase Record permite al programa convertir el contenido del registro y acceder a los datos por nombre de campo o por índice.

Tipos de datos

AS400DataType es una interfaz que define los métodos necesarios para la conversión de datos. Un programa Java utiliza los tipos de datos cuando es necesario convertir fragmentos de datos individuales. Existen clases de conversión para los siguientes tipos de datos:

- Numérico
- Texto (tipo carácter)
- Compuesto (numérico y texto)

Conversión que especifica un formato de registro

IBM Toolbox para Java proporciona clases sobre las que construir las clases de tipos de datos que permiten que los datos se conviertan de registro en registro, en vez de convertirse de campo en campo. Por ejemplo, supongamos que un programa Java lee datos de salida de una cola de datos. El objeto cola de datos devuelve al programa Java una matriz de bytes de datos de iSeries. Esta matriz puede, en potencia, contener muchos tipos de datos de iSeries. La aplicación puede convertir de uno en uno los campos de la matriz de bytes utilizando las clases de tipos de datos, o bien el programa puede crear un formato de registro que describa los campos en la matriz de bytes. Luego ese registro realiza la conversión.

La conversión de formato de registro puede ser de utilidad cuando se está trabajando con datos de la llamada a programa, de la cola de datos y de las clases de acceso a nivel de registro. La entrada y la salida de estas clases son matrices de bytes que pueden contener muchos campos de diversos tipos. Los conversores de formato de registro pueden hacer más fácil la conversión de estos datos entre el formato de iSeries y el formato Java.

La conversión mediante el formato de registro emplea tres clases:

- Las clases FieldDescription identifican un campo o parámetro con un tipo de datos y un nombre.
- Una clase RecordFormat describe un grupo de campos.
- Una clase Record une la descripción de un registro (en la clase RecordFormat) con los datos reales.
- Una clase LineDataRecordWriter escribe un registro en un objeto OutputStream en formato de datos de línea

Ejemplos

Dos ejemplos ilustran cómo se utilizan las clases de conversión de formato de registro con las colas de datos:

- Ejemplo: cómo se utilizan las clases Record y RecordFormat para poner datos en una cola
- Ejemplo: cómo se utilizan las clases FieldDescription, RecordFormat y Record

Clases de conversión para datos numéricos

Las clases de conversión para datos numéricos simplemente convierten los datos numéricos del formato utilizado en el servidor iSeries o AS/400e (denominado **formato del servidor** en la tabla siguiente) al formato Java. En la tabla siguiente se muestran los tipos de datos soportados:

Tipo numérico	Descripción
AS400Bin2	La conversión se realiza entre un número de dos bytes con signo con el formato del servidor y un objeto Short Java.
AS400Bin4	La conversión se realiza entre un número de cuatro bytes con signo con el formato del servidor y un objeto Integer Java.
AS400ByteArray	La conversión se realiza entre dos matrices de bytes. Es de utilidad porque el conversor rellena con ceros correctamente y rellena con datos el almacenamiento intermedio destino.
AS400Float4	La conversión se realiza entre un número de coma flotante de cuatro bytes con signo con el formato del servidor y un objeto Float Java.
AS400Float8	La conversión se realiza entre un número de coma flotante de ocho bytes con signo con el formato del servidor y un objeto Double Java.
AS400PackedDecimal	La conversión se realiza entre un número decimal empaquetado con el formato del servidor y un objeto BigDecimal Java.
AS400UnsignedBin2	La conversión se realiza entre un número de dos bytes sin signo con el formato del servidor y un objeto Integer Java.
AS400UnsignedBin4	La conversión se realiza entre un número de cuatro bytes sin signo con el formato del servidor y un objeto Long Java.
AS400ZonedDecimal	La conversión se realiza entre un número decimal con zona con el formato del servidor y un objeto BigDecimal Java.

En el ejemplo que sigue se muestra la conversión de un tipo numérico con el formato del servidor a un objeto int Java:

```
        // Cree un almacenamiento intermedio que contenga el
        // tipo de datos del servidor. Supongamos que el almacenamiento
        // intermedio se rellena con datos numéricos con el formato del
        // servidor procedentes de colas de datos, llamadas a programa,
etc.
     byte[] data = new byte[100];

        // Cree un conversor para este
        // tipo de datos del servidor.
       AS400Bin4 bin4Converter = new AS400Bin4();

        // Realice la conversión del tipo del servidor al
        // objeto Java. El número empieza al principio
        // del almacenamiento intermedio.
     Integer intObject = (Integer) bin4Converter.toObject(data,0);

        // Extraiga el tipo Java simple del
        // objeto Java.
     int i = intObject.intValue();
```

En el ejemplo que sigue se muestra la conversión de un objeto int Java a un tipo de datos numérico con el formato del servidor:

```
        // Cree un objeto Java que contenga
        // el valor que se ha de convertir.
     Integer intObject = new Integer(22);
```

```
    // Cree un conversor para el
    // tipo de datos del servidor.
   AS400Bin4 bin4Converter = new AS400Bin4();

    // Realice la conversión desde el objeto Java
    // al tipo de datos del servidor.
byte[] data = bin4Converter.toBytes(intObject);

    // Averigüe cuántos bytes del
    // almacenamiento intermedio se
    // rellenaron con el valor del servidor.
int length = bin4Converter.getByteLength();
```

Conversión de texto

Los datos de tipo carácter se convierten mediante la clase AS400Text. Esta clase hace que los datos de tipo carácter que tienen la página de códigos y el juego de caracteres (CCSID) EBCDIC se conviertan a Unicode. Al construirse el objeto AS400Text, el programa Java especifica la longitud de la serie que se debe convertir, así como el CCSID o la codificación del servidor. Se supone que el CCSID del programa Java es Unicode 13488. El método toBytes() hace que el formato Java se convierta en una matriz de bytes en formato de iSeries. El método toObject() hace que una matriz de bytes en formato iSeries se convierta en el formato Java.

La clase AS400BidiTransform proporciona transformaciones de diseño que hacen posible la conversión de texto bidireccional en formato de iSeries (tras convertirlo primero a Unicode) a texto bidireccional en formato Java, o de formato Java a formato de iSeries. La conversión por omisión se basa en el CCSID del trabajo. Para modificar la dirección y el formato del texto, especifique BidiStringType. Observe que cuando los objetos IBM Toolbox para Java llevan a cabo la conversión internamente, como en la clase DataArea, los objetos tienen un método para cambiar el tipo de serie. Por ejemplo, la clase DataArea tiene el método addVetoableChangeListener() que se puede especificar para escuchar cambios de veto a determinadas propiedades, entre ellas el tipo de serie.

Por ejemplo, supongamos que un objeto DataQueueEntry devuelve un texto iSeries en EBCDIC. En el ejemplo que sigue vemos cómo estos datos se convierten a Unicode para que el programa Java pueda utilizarlos:

```
          // ...Supongamos que ya se ha efectuado el trabajo de la cola
          // de datos para recuperar el texto del iSeries y los datos
          // se han colocado en el siguiente almacenamiento intermedio.
     int textLength = 100;
     byte[] data = new byte[textLength];

          // Cree un conversor para el tipo de datos iSeries. Observe que se
construye
          // un conversor por omisión. Éste supone que la página de códigos
EBCDIC de iSeries
          // coincide con el entorno nacional del cliente. Si ello no
corresponde a la
          // realidad, el programa Java puede especificar explícitamente el
CCSID de
          // EBCDIC que se debe utilizar. Sin embargo, se recomienda
especificar un
          // CCSID cuando sea posible (consulte las notas siguientes).
     AS400Text textConverter = new AS400Text(textLength)

          // Nota: si lo desea, puede crear un conversor para un CCSID
          // específico. Utilice un objeto AS400 por si el programa se
ejecuta
          // como un cliente proxy de Toolbox para Java.
     int ccsid = 37;
     AS400 system = ...; // Objeto AS400
     AS400Text textConverter = new AS400Text(textLength, ccsid, system);

          // Nota: también puede crear un conversor sólo con el objeto
AS400.
          // Este conversor supone que la página de códigos de iSeries
          // coincide con el CCSID devuelto por el objeto AS400.
     AS400Text textConverter = new AS400Text(textLength, system);

          // Convierta los datos de EBCDIC a Unicode. Si la longitud
          // del objeto AS400Text es superior al número de caracteres
          // convertidos, la serie (String) resultante se rellenará
```

```
        // con blancos hasta la longitud especificada.
    String javaText = (String) textConverter.toObject(data);
```

«

Clases de conversión para tipos compuestos

Las clases de conversión para tipos compuestos son las siguientes:

- AS400Array - Permite al programa Java trabajar con una matriz de tipos de datos.
- AS400Structure - Permite al programa Java trabajar con una estructura cuyos elementos son tipos de datos.

En el ejemplo que sigue se muestra la conversión desde una estructura Java a una matriz de bytes, y cómo se realiza la conversión inversa. El ejemplo presupone que se emplea el mismo formato de datos tanto para el envío como para la recepción de los datos.

```
        // Cree una estructura de tipos de datos
        // que se corresponda con una estructura
        // que contiene:
        //    - un número de cuatro bytes
        //    - cuatro bytes de relleno
        //    - un número de ocho bytes
        //    - 40 caracteres
     AS400DataType[] myStruct =
     {
        new AS400Bin4(),
        new AS400ByteArray(4),
        new AS400Float8(),
        new AS400Text(40)
     };

        // Cree un objeto conversión que
        // utilice la estructura.
     AS400Structure myConverter = new AS400Structure(myStruct);

        // Cree el objeto Java que contiene
        // los datos que deben enviarse al servidor.
     Object[] myData =
     {
          new Integer(88),        // el número de cuatro bytes
          new byte[0],            // el relleno (permite rellenar con 0 el
objeto conversión)
          new Double(23.45),      // el número de coma flotante de ocho
bytes
          "Esa es mi estructura"  // la serie de caracteres
     };

        // Haga que el objeto Java se convierta en matriz de bytes.
     byte[] myAS400Data = myConverter.toBytes(myData);


        // ...Envíe la matriz de bytes al servidor.
        // Obtenga los datos nuevamente del servidor.
        // Los datos devueltos serán asimismo una
        // matriz de bytes.


        // Haga que los datos devueltos del
        // iSeries se conviertan al formato Java.
     Object[] myRoundTripData =
                 (Object[])myConverter.toObject(myAS400Data,0);

        // Saque el tercer objeto de la
```

```
    // estructura. Es el objeto double.
Double doubleObject = (Double) myRoundTripData[2];

    // Extraiga el tipo Java simple del
    // objeto Java.
double d = doubleObject.doubleValue();
```

Clases de descripción de campo

Las clases de descripciones de campo permiten al programa Java describir el contenido de un campo o un parámetro con un tipo de datos y una serie que contenga el nombre del campo. El programa, si está trabajando con datos de acceso a nivel de registro, puede asimismo especificar cualquier palabra clave de especificación de definición de datos (DDS) iSeries o AS/400e que describa el campo con más detalle.

Las clases de descripción de campo son las siguientes:

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

Por ejemplo, supongamos que las entradas existentes en una cola de datos tienen el mismo formato. Cada entrada tiene un número de mensaje (AS400Bin4), una indicación de la hora (8 caracteres) y un texto de mensaje (50 caracteres). Las descripciones de campo correspondientes podrían ser estas:

```
         // Cree una descripción de campo para
         // los datos numéricos. Observe que utiliza
         // el tipo de datos AS400Bin4. También se
         // denomina el campo para que sea posible
         // acceder a él por el nombre en la
         // clase registro.
     BinaryFieldDescription bfd = new BinaryFieldDescription(new
AS400Bin4(),
                                                             "msgNumber");

         // Cree una descripción de campo para los
         // datos de tipo carácter. Observe que utiliza
         // el tipo de datos AS400Text. También se
         // denomina el campo para que sea posible
         // acceder a él por el nombre en la
         // clase registro.
     CharacterFieldDescription cfd1 = new CharacterFieldDescription(new
AS400Text(8),

"msgTime");

         // Cree una descripción de campo para los
         // datos de tipo carácter. Observe que utiliza
         // el tipo de datos AS400Text. También se
```

```
        // denomina el campo para que sea posible
        // acceder a él por el nombre en la
        // clase registro.
     CharacterFieldDescription cfd2 = new CharacterFieldDescription(new
AS400Text(50),

"msgText");
```

Ahora las descripciones de campo ya están listas para agruparse en una clase formato de registro. El ejemplo continúa en la sección Formato de registro.

Clase RecordFormat

La clase RecordFormat permite al programa Java describir un grupo de campos o parámetros. Un objeto Record contiene los datos descritos por un objeto RecordFormat. Si el programa utiliza clases de acceso a nivel de registro, la clase RecordFormat también permite al programa especificar descripciones para campos clave.

Un objeto RecordFormat contiene un conjunto de descripciones de campo. El acceso a la descripción de campo se puede realizar por índice o por nombre. En la clase RecordFormat hay métodos que permiten realizar estas tareas:

- Añadir descripciones de campo al formato de registro.
- Añadir descripciones de campo clave al formato de registro.
- Recuperar descripciones de campo del formato de registro por índice o por nombre.
- Recuperar descripciones de campo clave del formato de registro por índice o por nombre.
- Recuperar los nombres de los campos que constituyen el formato de registro.
- Recuperar los nombres de los campos clave que constituyen el formato de registro.
- Recuperar el número de campos existentes en el formato de registro.
- Recuperar el número de campos clave existentes en el formato de registro.
- Crear un objeto Record basado en este formato de registro.

Por ejemplo, para añadir a un formato de registro las descripciones de campo creadas en el ejemplo descripción de campo:

```
        // Cree un objeto formato de registro;
        // después rellénelo con descripciones
        // de campo.
    RecordFormat rf = new RecordFormat();
    rf.addFieldDescription(bfd);
    rf.addFieldDescription(cfd1);
    rf.addFieldDescription(cfd2);
```

Ahora el programa ya está listo para crear un registro a partir del formato de registro. El ejemplo continúa en la sección Registro.

Clase Record

La clase de registro permite al programa Java procesar los datos descritos por la clase de formato de registro. La conversión de datos se realiza entre matrices de bytes que contienen datos del servidor y objetos Java. En la clase registro hay métodos que permiten realizar estas tareas:

- Recuperar el contenido de un campo, por índice o por nombre, como objeto Java.
- Recuperar el número de campos del registro.
- Establecer el contenido de un campo, por índice o por nombre, con un objeto Java.
- Recuperar el contenido del registro como datos del servidor en una matriz de bytes o en una corriente de datos de salida.
- Establecer el contenido del registro desde una matriz de bytes o una corriente de datos de entrada.
- Convertir el contenido del registro en una serie (String).

Por ejemplo, para utilizar el formato de registro creado en el ejemplo de formato de registro:

```
    // Supongamos que ya se ha realizado el trabajo
    // de configuración de la cola de datos. Ahora
    // debe leerse un registro de la cola de datos.
DataQueueEntry dqe = dq.read();

    // Los datos de la cola de datos están ahora
    // en una entrada de cola de datos. Obtenga
    // los datos de la entrada de cola de datos
    // y póngalos en el registro.
    // Obtenemos un registro por omisión del
    // objeto formato de registro y lo
    // inicializamos con los datos de la
    // entrada de cola de datos.
Record dqRecord = rf.getNewRecord(dqe.getData());

    // Ahora que los datos están en el registro,
    // saque los datos de un campo, a razón de
    // un campo cada vez, convirtiendo los datos
    // a medida que se vayan eliminando. El resultado
    // son los datos de un objeto Java que, ahora,
    // el programa puede procesar.
Integer msgNumber = (Integer) dqRecord.getField("msgNumber");
String  msgTime   = (String)  dqRecord.getField("msgTime");
String  msgText   = (String)  dqRecord.getField("msgText");
```

Clase LineDataRecordWriter

La clase LineDataRecordWriter escribe los datos de registro, en formato de datos de línea, en un objeto OutputStream. La clase convierte los datos en bytes utilizando el CCSID especificado. El formato de registro asociado al registro determina el formato de los datos.

La utilización de LineDataRecordWriter requiere que se hayan establecido los siguientes atributos de formato de registro:

- ID de formato de registro
- Tipo de formato de registro

Junto con las clases Record o RecordFormat, la clase LineDataRecordWriter toma un registro como entrada para el método writeRecord(). (El registro toma RecordFormat como entrada cuando se crea una instancia del mismo.)

La clase LineDataRecordWriter proporciona métodos que permiten llevar a cabo estas acciones:

- Obtener el CCSID
- Obtener el nombre de la codificación
- Escribir los datos de registro, en formato de datos de línea, en un objeto OutputStream

Ejemplo: cómo se utiliza la clase LineDataRecordWriter

```
     // Ejemplo de cómo se utiliza la clase LineDataRecordWriter.
                           try
        {
        // Cree un CCSID
        ccsid_ = system_.getCcsid();

        // Cree una cola de salida e indique que los datos del archivo en
spool sean *LINE
          OutputQueue outQ = new OutputQueue(system_,
"/QSYS.LIB/RLPLIB.LIB/LDRW.OUTQ");
            PrintParameterList parms = new PrintParameterList();
        parms.setParameter(PrintObject.ATTR_PRTDEVTYPE, "*LINE");

        // Inicialice el formato de registro para escribir datos
        RecordFormat recfmt = initializeRecordFormat();

        // Cree un registro y asigne datos para imprimir...
        Record record = new Record(recfmt);
        createRecord(record);

        SpooledFileOutputStream os = null;
  try {
            // Cree el archivo en spool de salida para contener los datos de
registro
            os = new SpooledFileOutputStream(system_, parms, null, outQ);
          }
    catch (Exception e) {
               System.out.println("Se ha producido un error al crear el
archivo en spool");
      e.printStackTrace();
          }

                // Cree el transcriptor de registro de datos de línea
```

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

      // Escriba el registro de datos
            ldw.writeRecord(record);

      // Cierre el objeto OutputStream
os.close();
    }

    catch(Exception e)
    {
      failed(e, "Se ha producido una excepción.");
    }
```

Colas de datos

Las clases DataQueue permiten al programa Java interactuar con las colas de datos del servidor. Las colas de datos de los servidores iSeries y AS/400e tienen las características siguientes:

- La cola de datos permite agilizar las comunicaciones entre trabajos. Por lo tanto, es un excelente procedimiento para sincronizar los trabajos y pasar datos entre ellos.
- Muchos trabajos pueden acceder simultáneamente a las colas de datos.
- Los mensajes pueden tener un formato libre en una cola de datos. Los campos no son obligatorios como en los archivos de base de datos.
- La cola de datos puede utilizarse tanto para procesos síncronos como asíncronos.
- En una cola de datos, los mensajes pueden ordenarse de varias maneras:
  - Último en entrar, primero en salir (LIFO). El último mensaje (el más reciente) colocado en la cola de datos es el primer mensaje que se saca de ella.
  - Primero en entrar, primero en salir (FIFO). El primer mensaje (el más antiguo) colocado en la cola de datos es el primer mensaje que se saca de ella.
  - Por clave. Cada mensaje de la cola de datos tiene una clave asociada a él. Para poder sacar un mensaje de la cola de datos, es preciso especificar la clave asociada al mismo.

Las clases de cola de datos proporcionan un conjunto completo de interfaces que permiten acceder a las colas de datos del servidor desde el programa Java. Es un excelente método de comunicación entre los programas Java y los programas del servidor escritos en cualquier lenguaje de programación.

Cada objeto de cola de datos tiene como parámetro obligatorio el objeto AS400 que representa el servidor que tiene la cola de datos o donde se debe crear la cola de datos.

La utilización de las clases de cola de datos hace que el objeto AS400 se conecte al servidor. En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones.

Cada objeto cola de datos requiere el nombre de vía de acceso del sistema de archivos integrado de la cola de datos. El tipo correspondiente a la cola de datos es DTAQ. En Nombres de vía de acceso del sistema de archivos integrado encontrará más información.

Colas de datos secuenciales y por clave

Las clases de cola de datos dan soporte a las colas de datos siguientes:

- Colas de datos secuenciales
- Colas de datos por clave

Los métodos comunes a los dos tipos de colas están en la clase BaseDataQueue. La clase DataQueue amplía la clase BaseDataQueue para completar la implementación de las colas de datos secuenciales. La clase KeyedDataQueue amplía la clase BaseDataQueue para completar la implementación de las colas de datos por clave.

Los datos, cuando se leen de una cola de datos, se colocan en un objeto DataQueueEntry. Este objeto contiene los datos de los dos tipos de colas de datos, las que son por clave y las secuenciales. Los datos adicionales disponibles cuando se leen de una cola de datos por clave se colocan en un objeto KeyedDataQueueEntry que amplía la clase DataQueueEntry.

Las clases de cola de datos no alteran los datos que se escriben en la cola de datos del servidor o se leen de ella. El programa Java debe dar el formato correcto a los datos. Las clases de conversión de datos proporcionan métodos para convertir los datos.

En el ejemplo que sigue se crea un objeto DataQueue, se leen datos del objeto DataQueueEntry y luego se lleva a

cabo la desconexión del sistema.

```
        // Cree un objeto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

        // Cree el objeto DataQueue.
     DataQueue dq = new DataQueue(sys, "/QSYS.LIB/MYLIB.LIB/MYQUEUE.DTAQ");

        // Lea datos de la cola.
     DataQueueEntry dqData = dq.read();

        // Obtenga datos del objeto DataQueueEntry.
     byte[] data = dqData.getData();

        // ...Procese los datos.

        // Desconecte, puesto que ya ha terminado de utilizar las colas de
datos.
     sys.disconnectService(AS400.DATAQUEUE);
```

Colas de datos secuenciales

Las entradas existentes en una cola de datos secuencial del servidor se eliminan por orden FIFO (primero en entrar, primero en salir) o LIFO (último en entrar, primero en salir). Las clases BaseDataQueue y DataQueue proporcionan los siguientes métodos para trabajar con las colas de datos secuenciales:

- El método create, que permite crear una cola de datos en el servidor. El programa Java debe especificar el tamaño máximo de una entrada en la cola de datos. El programa Java puede especificar opcionalmente parámetros de cola de datos (FIFO frente a LIFO, guardar información de remitente, especificar información de autorización, forzar en disco y proporcionar una descripción de cola) cuando se crea la cola.
- El método peek at, que permite echar una mirada rápida a una entrada en la cola de datos sin eliminar dicha entrada de la cola.El programa Java puede esperar o devolver inmediatamente si en este momento no hay ninguna entrada en la cola.
- El método read, que permite leer una entrada quitándola de la cola.El programa Java puede esperar o devolver inmediatamente si no hay ninguna entrada disponible en la cola.
- El método write, que permite escribir una entrada en la cola.
- El método clear, que permite borrar todas las entradas de la cola.
- El método delete, que permite suprimir la cola.

La clase BaseDataQueue proporciona métodos adicionales para recuperar los atributos de la cola de datos.

Ejemplos

Ejemplos de cola de datos secuencial, en los que el productor pone elementos en una cola de datos y el consumidor saca elementos de la cola de datos y los procesa:

- Ejemplo de productor de cola de datos secuencial.
- Ejemplo de consumidor de cola de datos secuencial.

Colas de datos por clave

Las clases BaseDataQueue y KeyedDataQueue proporcionan los siguientes métodos para trabajar con colas de datos por clave:

- El método create, que permite crear una cola de datos por clave en el servidor. El programa Java debe especificar la longitud de la clave y el tamaño máximo de una entrada en la cola. El programa Java puede especificar opcionalmente información de autorización, guardar información de remitente, forzar en disco y proporcionar una descripción de cola.
- El método peek, que permite echar una mirada rápida a una entrada (en función de la clave especificada) sin eliminar la entrada de la cola. El programa Java puede esperar o devolver inmediatamente si en este momento no hay en la cola ninguna entrada que coincida con los criterios de la clave.
- El método read, que permite leer una entrada (en función de la clave especificada) quitándola de la cola. El programa Java puede esperar o devolver inmediatamente, si en la cola no hay ninguna entrada disponible que coincida con los criterios de la clave.
- El método write, que permite escribir una entrada por clave en la cola.
- El método clear, que permite borrar todas las entradas o sólo las que coincidan con una clave especificada.
- El método delete, que permite suprimir la cola.

Las clases BaseDataQueue y KeyedDataQueue proporcionan asimismo métodos adicionales que permiten recuperar los atributos de la cola de datos.

Ejemplos

En los siguientes ejemplos de cola de datos por clave, el productor pone elementos en una cola de datos y el consumidor saca elementos de la cola y los procesa:

- Ejemplo de productor de cola de datos por clave
- Ejemplo de consumidor de cola de datos por clave

Certificados digitales

Los certificados digitales son sentencias con firma digital utilizadas para las transacciones protegidas en Internet. (Los certificados digitales pueden emplearse en servidores que ejecuten una versión de OS/400 igual o posterior a la Versión 4 Release 3 (V4R3).) Para establecer una conexión segura utilizando SSL (capa de sockets segura), se requiere un certificado digital.

Los certificados digitales constan de estos elementos:

- La clave pública de cifrado del usuario
- El nombre y la dirección del usuario
- La firma digital de una autoridad certificadora (CA) de terceros. La firma de la autoridad garantiza que el usuario es una entidad de confianza.
- La fecha de emisión del certificado
- La fecha de caducidad del certificado

Como administrador de un servidor protegido, puede añadir al servidor una "clave de raíz de confianza" de una autoridad certificadora. Esto quiere decir que el servidor se fiará de cualquier persona que disponga de un certificado procedente de esa determinada autoridad certificadora.

Los certificados digitales ofrecen también el cifrado, garantizando una transferencia segura de los datos mediante una clave privada de cifrado.

Es posible crear certificados digitales con la herramienta javakey. (Encontrará más información sobre javakey y la seguridad Java en la página sobre seguridad Java de Sun Microsystems, Inc.  .) El programa bajo licencia IBM Toolbox para Java tiene clases que administran certificados digitales en el servidor iSeries o AS/400e.

Las clases AS400Certificate proporcionan métodos para gestionar certificados X.509 ASN.1 codificados. Se proporcionan clases para realizar estas tareas:

- Obtener y establecer datos de certificados.
- Listar certificados por lista de validación o perfil de usuario.
- Gestionar certificados; por ejemplo, añadir un certificado a un perfil de usuario o suprimir un certificado de una lista de validación.

La utilización de una clase de certificado hace que el objeto AS400 se conecte al servidor. En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones.

En el servidor, los certificados pertenecen a una lista de validación o a un perfil de usuario.

- La clase AS400CertificateUserProfileUtil tiene métodos para gestionar certificados en un perfil de usuario.
- La clase AS400CertificateVldlUtil tiene métodos para gestionar los certificados de una lista de validación.

Estas dos clases amplían la clase AS400CertificateUtil, que es una clase base abstracta que define métodos que son comunes a las dos subclases.

La clase AS400Certificate proporciona métodos para leer y escribir datos de certificado. El acceso a los datos se realiza en forma de matriz de bytes. El paquete Java.Security de la máquina virtual Java 1.2 proporciona clases que permiten obtener y establecer campos individuales del certificado.

Lista de certificados

Para obtener una lista de certificados, el programa Java debe llevar a cabo estas tareas:

1. Crear un objeto AS400.
2. Construir el objeto certificado correcto. Los objetos que se utilizan para listar certificados en un perfil de

usuario (AS400CertificateUserProfileUtil) son distintos de los utilizados para listar certificados en una lista de validación (AS400CertificateVldlUtil).

3. Crear criterios de selección basados en los atributos de certificado. La clase AS400CertificateAttribute contiene atributos que se utilizan como criterios de selección. Uno o varios objetos atributo definen los criterios que deben satisfacerse para poder añadir un certificado a la lista. Por ejemplo, una lista podría contener únicamente certificados para un determinado usuario u organización.
4. Crear un espacio de usuario en el servidor y poner el certificado en el espacio de usuario. Una operación de listar puede generar una gran cantidad de datos. Para que un programa Java pueda recuperar los datos, primero es necesario ponerlos en un espacio de usuario. El método listCertificates() permite poner los certificados en el espacio de usuario.
5. El método getCertificates() permite recuperar certificados del espacio de usuario.

El ejemplo que figura a continuación lista certificados de una lista de validación. Solo lista los certificados pertenecientes a una determinada persona.

```
        // Cree un objeto AS400. Los
        // certificados están en este sistema.
    AS400 sys = new AS400("mySystem.myCompany.com");


        // Cree el objeto certificado.
    AS400CertificateVldlUtil certificateList =
             new AS400CertificateVldlUtil(sys,
"/QSYS.LIB/MYLIB.LIB/CERTLIST.VLDL");


        // Cree la lista de atributos de certificado.
        // Solo queremos certificados correspondientes
        // a una única persona, por lo que la lista
        // consta de un solo elemento.
    AS400CertificateAttribute[] attributeList = new
AS400CertificateAttribute[1];
    attributeList[0] = new
AS400CertificateAttribute(AS400CertificateAttribute.SUBJECT_COMMON_NAME,
"Jane Doe");


        // Recupere la lista que coincide con
        // los criterios. Se utilizará el espacio
        // de usuario "myspace" de la biblioteca
        // "mylib" para almacenar los certificados.
        // El espacio de usuario debe existir antes de
        // llamar a esta API.
    int count = certificateList.listCertificates(attributeList,
           "/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");


        // Recupere los certificados
        // del espacio de usuario.
    AS400Certificates[] certificates =
certificateList.getCertificates("/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC", 0, 8);


        // ...Procese los certificados.
```

Clase EnvironmentVariable

La clase EnvironmentVariable y la clase EnvironmentVariableList permiten acceder a variables de entorno **a nivel de sistema** de iSeries y establecerlas.

Cada una de las variables tiene identificadores exclusivos: el nombre de sistema y el nombre de variable de entorno. Cada variable de entorno tiene asociado un CCSID (que por omisión es el CCSID del trabajo actual) que describe dónde está almacenado el contenido de la variable.

Nota: las variables de entorno son distintas de los valores del sistema, aunque a menudo se utilizan con el mismo fin. Encontrará más información acerca de cómo acceder a los valores del sistema en Valores del sistema.

Utilice un objeto EnvironmentVariable para llevar a cabo las acciones siguientes en una variable de entorno:

- Obtener y establecer el nombre
- Obtener y establecer el sistema
- Obtener y establecer el valor (lo que permite cambiar el CCSID)
- Renovar el valor

Ejemplo: crear, establecer y obtener variables de entorno

El ejemplo que sigue crea dos variables de entorno (EnvironmentVariable) y posteriormente establece y obtiene sus valores.

```
     // Cree el objeto de sistema iSeries.
     AS400 system = new AS400("mySystem");
     // Cree la variable de entorno de color de primer plano y establézcala
en rojo ("red").
     EnvironmentVariable fg = new EnvironmentVariable(system, "FOREGROUND");
     fg.setValue("RED");
     // Cree la variable de entorno de color de fondo y obtenga su valor.
     EnvironmentVariable bg = new EnvironmentVariable(system, "BACKGROUND");
     String background = bg.getValue();
```

Excepciones

Las clases de acceso de IBM Toolbox para Java lanzan excepciones cuando se producen errores de dispositivo, limitaciones físicas, errores de programación o errores de entrada de usuario. Las clases de excepción se basan en el tipo de error que se produce, en vez de basarse en la ubicación origen del error.

La mayor parte de las excepciones contiene tres fragmentos informativos:

- **Tipo de error** - El objeto excepción lanzado indica qué tipo de error se ha producido. Los errores del mismo tipo se agrupan en una clase de excepción.
- **Detalles del error** - La excepción contiene un código de retorno para identificar con más precisión la causa del error producido. Los valores del código de retorno son constantes dentro de la clase de excepción.
- **Texto del error** - La excepción contiene una serie de caracteres descriptiva del error que se ha producido. La serie se traduce al idioma del entorno nacional de la máquina virtual Java (JVM) del cliente.

El siguiente ejemplo muestra cómo capturar una excepción lanzada, recuperar el código de retorno y visualizar el texto de la excepción:

```
     // ...Ya se ha terminado todo el trabajo de
     // preparación para suprimir un archivo en
     // el servidor mediante la clase IFSFile.
     // Ahora intente suprimir el archivo.
                       try
{
   aFile.delete();

}

     // La supresión ha fallado.
catch (ExtendedIOException e)
{
     // Visualice la serie traducida
     // que indica la razón por la que
     // ha fallado la supresión.
   System.out.println(e);

     // Obtenga el código de retorno de la
     // excepción y visualice información
     // adicional basada en el código de
     // retorno.
   int rc = e.getReturnCode()

   switch (rc)
   {
      case ExtendedIOException.FILE_IN_USE:
         System.out.println("Supresión anómala, archivo en uso "):
         break;

      case ExtendedIOException.PATH_NOT_FOUND:
         System.out.println("Supresión anómala, vía no encontrada ");
         break;

     // ...Para cada error específico cuyo
     // seguimiento desea efectuar.

      default:
         System.out.println("Supresión anómala, rc = ");
         System.out.println(rc);
```

```
    }
}
```

Clase FTP

La clase FTP proporciona una interfaz programable con las funciones FTP. Ya no es necesario que utilice java.runtime.exec() ni que indique a los usuarios que ejecuten los mandatos FTP en una aplicación aparte. Es decir, puede programar las funciones FTP directamente en la aplicación. Así, desde dentro de su programa, puede hacer estas tareas:

- Conectarse a un servidor FTP
- Enviar mandatos al servidor
- Listar los archivos de un directorio
- Obtener archivos del servidor **y**
- Poner archivos en el servidor

Por ejemplo, con la clase FTP, puede copiar en un servidor un conjunto de archivos de un directorio:

```
     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("Copiando " + entries[i]);
                            try
       {
         client.get(entries[i], "c:\\ftptest\\" + entries[i]);
       }
        catch (Exception e)
       {
         System.out.println("  la operación de copia ha fallado;
probablemente sea un directorio");
       }
     }

     client.disconnect();
```

FTP es una interfaz genérica que funciona con numerosos y variados servidores FTP. Por lo tanto, es responsabilidad del programador el adoptar la semántica del servidor.

Subclase de FTP

Mientras que la clase FTP es una interfaz FTP genérica, la subclase AS400FTP está escrita específicamente para el servidor FTP del servidor. Es decir, esta subclase entiende la semántica del servidor FTP en el servidor iSeries o AS/400e, de modo que el programador no tiene que preocuparse de ello. Por ejemplo, esta clase entiende los diversos pasos que se necesitan para transferir al servidor un archivo de salvar y los lleva a cabo automáticamente. AS400FTP también encaja bien en los recursos de seguridad de IBM Toolbox para Java. Al igual que con las otras clases de IBM Toolbox para Java, AS400FTP depende del objeto AS400 para obtener el nombre del sistema, el ID de usuario y la contraseña.

El ejemplo siguiente pone un archivo de salvar en el servidor. Fíjese en cómo la aplicación no establece el tipo de transferencia de datos en binario ni utiliza CommandCall de la Caja de Herramientas para crear el archivo de salvar. Debido a que la extensión es .savf, la clase AS400FTP detecta que el archivo que se ha de poner es un archivo de salvar y realiza esos pasos automáticamente.

```
AS400 system = new AS400();
 AS400FTP ftp = new AS400FTP(system);
```

```
ftp.put("myData.savf", "/QSYS.LIB/MYLIB.LIB/MYDATA.SAVF");
```

Sistema de archivos integrado

Las clases del sistema de archivos integrado permiten a un programa Java acceder a los archivos del sistema de archivos integrado de un servidor iSeries o AS/400e como si fuesen una corriente de bytes o una corriente de caracteres. Las clases del sistema de archivos integrado se crearon debido a que el paquete java.io no proporciona la función de redirección de archivos ni otras funciones del iSeries.

Las funciones proporcionadas por las clases IFSFile forman un superconjunto de las funciones proporcionadas por las clases de entrada/salida (IO) de archivo existentes en el paquete java.io. Todos los métodos de java.io FileInputStream, FileOutputStream y RandomAccessFile están en las clases del sistema de archivos integrado.

Además de tales métodos, las clases contienen métodos para llevar a cabo las tareas siguientes:

- Especificar una modalidad de compartimiento de archivos para denegar el acceso a un archivo que se esté utilizando
- Especificar una modalidad de creación de archivo para abrir, crear o sustituir el archivo
- Bloquear una sección del archivo y denegar el acceso a dicha parte del archivo mientras se esté utilizando
- Listar el contenido de un directorio de manera más eficaz
- Almacenar en la antememoria el contenido de un directorio para mejorar el rendimiento al limitar las llamadas al servidor
- Determinar el número de bytes que están disponibles en el sistema de archivos del servidor
- Permitir a un applet Java acceder a los archivos del sistema de archivos del servidor
- Leer y escribir datos como texto, en vez de como datos binarios
- Determinar el tipo de objeto archivo (lógico, físico, de salvar, etc.) cuando el objeto está en el sistema de archivos QSYS.LIB

Mediante las clases del sistema de archivos integrado, el programa Java puede acceder directamente a los archivos continuos existentes en el iSeries. El programa Java puede seguir utilizando el paquete java.io, pero en ese caso el sistema operativo del cliente debe proporcionar un método de redirección. Por ejemplo, si el programa Java se está ejecutando en un sistema operativo Windows 95 o Windows NT, se requiere la función Unidades de red de iSeries Access para Windows a fin de redirigir las llamadas java.io al iSeries. Con las clases del sistema de archivos integrado, no se necesita iSeries Access para Windows.

Las clases del sistema de archivos integrado tienen como parámetro obligatorio el objeto AS400 que representa el sistema iSeries que contiene el archivo. El hecho de utilizar las clases de sistema de archivos integrado hace que el objeto AS400 se conecte al iSeries. En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones.

Las clases del sistema de archivos integrado requieren el nombre jerárquico que el objeto tiene en el sistema de archivos integrado. Como carácter separador de las vías de acceso, utilice la barra inclinada hacia delante. El ejemplo que figura a continuación muestra cómo se accede a FILE1, en la vía de directorio DIR1/DIR2:

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

Las clases del sistema de archivos integrado son:

Clase del sistema de archivos integrado	Descripción
IFSFile	Representa un archivo del sistema de archivos integrado
IFSJavaFile	Representa un archivo del sistema de archivos integrado (amplía java.io.File)
IFSFileInputStream	Representa una corriente de entrada para leer datos de un archivo iSeries
IFSTextFileInputStream	Representa una corriente de datos de tipo carácter leídos en un archivo

IFSFileOutputStream	Representa una corriente de salida para escribir datos en un archivo iSeries
IFSTextFileOutputStream	Representa una corriente de datos de tipo carácter que se están escribiendo en un archivo
IFSRandomAccessFile	Representa un archivo en el iSeries para leer y escribir datos
IFSFileDialog	Permite al usuario moverse dentro del sistema de archivos y seleccionar un archivo del sistema de archivos

Ejemplos

El ejemplo de IFSCopyFile muestra cómo se utilizan las clases del sistema de archivos integrado para copiar un archivo de un directorio en otro en el iSeries.

El ejemplo de lista de archivos muestra cómo se utilizan las clases del sistema de archivos integrado para listar el contenido de un directorio en el iSeries.

Clase IFSFile

La clase IFSFile representa un objeto del sistema de archivos integrado del iSeries. Los métodos que hay en IFSFile representan operaciones realizadas en el objeto como un todo. Se puede utilizar IFSFileInputStream, IFSFileOutputStream y IFSRandomAccessFile para leer y escribir en el archivo. La clase IFSFile permite al programa Java llevar a cabo estas tareas:

- Determinar si el objeto existe y es un directorio o un archivo
- Determinar si el programa Java puede leer en un archivo o escribir en él
- Determinar la longitud de un archivo
- Determinar los permisos de un objeto y establecer los permisos de un objeto
- Crear un directorio
- Suprimir un archivo o un directorio
- Redenominar un archivo o un directorio
- Obtener o establecer la fecha de la última modificación de un archivo
- Listar el contenido de un directorio
- Listar el contenido de un directorio y guardar la información de atributos en una antememoria local
- Determinar la cantidad de espacio disponible en el sistema
- Determinar el tipo de objeto archivo cuando está en el sistema de archivos QSYS.LIB

Puede obtener la lista de archivos de un directorio mediante el método list() o el método listFiles():

- El método listFiles() almacena en la antememoria información de cada uno de los archivos de la llamada inicial. Tras efectuar la llamada a listFiles(), la utilización de otros métodos para consultar detalles de archivo permite obtener un mejor rendimiento ya que la información se recupera de la antememoria. Por ejemplo, realizar una llamada a isDirectory() en un objeto IFSFile devuelto por listFiles() no requiere efectuar una llamada al servidor.
- El método list() recupera información sobre cada uno de los archivos en una petición independiente realizada al servidor, lo que hace que sea más lento y que utilice más recursos del servidor.

Nota: la utilización de listFiles() significa que la información de la antememoria puede quedar obsoleta, por lo que puede que deba renovar los datos volviendo a efectuar una llamada a listFiles().

Ejemplos

Los ejemplos que hay a continuación muestran cómo se utiliza la clase IFSFile:

- **Ejemplo:** crear un directorio
- **Ejemplo:** cómo se utilizan las excepciones para hacer seguimiento de errores
- **Ejemplo:** listar archivos con la extensión .txt
- **Ejemplo:** cómo se utiliza listFiles() para listar el contenido de un directorio

Clase IFSJavaFile

La clase IFSJavaFile representa un archivo del sistema de archivos integrado de iSeries y amplía la clase java.io.File. IFSJavaFile permite escribir archivos para la interfaz java.io.File que accede a los sistemas de archivos integrados de iSeries.

IFSJavaFile crea interfaces portables que son compatibles con java.io.File y únicamente utiliza los errores y las excepciones que utiliza la clase java.io.File. IFSJavaFile emplea las características del gestor de seguridad de java.io.File; pero, a diferencia de java.io.File, IFSJavaFile emplea las características de seguridad de forma continua.

La clase IFSJavaFile se utiliza junto con IFSFileInputStream y IFSFileOutputStream. No da soporte a las clases java.io.FileInputStream y java.io.FileOutputStream.

IFSJavaFile está basada en IFSFile; sin embargo, la interfaz de IFSJavaFile se parece más a java.io.File que a IFSFile. IFSFile es una alternativa de la clase IFSJavaFile.

Puede obtener la lista de archivos de un directorio con el método list() o el método listFiles():

- El método listFiles() permite obtener un mejor rendimiento ya que recupera y almacena en la antememoria la información de cada uno de los archivos de la llamada inicial. Posteriormente, la información de cada uno de los archivos se recupera de la antememoria.
- El método list() recupera información sobre cada uno de los archivos en una petición independiente, lo que hace que sea más lento y que utilice más recursos del servidor.

Nota: la utilización de listFiles() significa que la información de la antememoria puede quedar obsoleta, por lo que puede que deba renovar los datos.

Más abajo figura un ejemplo de cómo se utiliza la clase IFSJavaFile.

```
// Trabaje con /Dir/File.txt en la memoria flash del sistema.
AS400 as400 = new AS400("flash");
IFSJavaFile file = new IFSJavaFile(as400, "/Dir/File.txt");


// Determine el directorio padre del archivo.
String directory = file.getParent();


// Determine el nombre del archivo.
String name = file.getName();


// Determine el tamaño del archivo.
long length = file.length();


// Determine cuándo se modificó el archivo por última vez.
Date date = new Date(file.lastModified());


   // Suprima el archivo.
if (file.delete() == false)
{
  // Visualice el código de error.
  System.err.println("No se ha podido suprimir el archivo.");
}
```

```
                          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 ("Excepción: " + e.getMessage());
}
```

IFSFileInputStream

La clase IFSFileInputStream representa una corriente de entrada para leer datos de un archivo en el servidor. Al igual que en la clase IFSFile, en IFSFileInputStream hay métodos que duplican los métodos de FileInputStream del paquete java.io. Además de estos métodos, IFSFileInputStream dispone de métodos adicionales específicos para los servidores iSeries y AS/400e. La clase IFSFileInputStream permite a un programa Java realizar estas tareas:

- Abrir un archivo para lectura. El archivo debe existir debido a que esta clase no crea archivos en el servidor. Puede utilizar un constructor que le permita especificar la modalidad de compartimiento de archivo.
- Determinar el número de bytes de la corriente.
- Leer bytes de la corriente.
- Saltarse bytes de la corriente.
- Bloquear o desbloquear bytes de la corriente.
- Cerrar el archivo.

Al igual que la clase FileInputStream de java.io, esta clase permite a un programa Java leer una corriente de bytes del archivo. El programa Java lee los bytes de modo secuencial con la única opción adicional de saltarse bytes de la corriente.

El ejemplo que sigue muestra cómo se utiliza la clase IFSFileInputStream.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Abra un objeto archivo que
    // represente el archivo.
IFSFileInputStream aFile =
              new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");

    // Determine el número de bytes
    // del archivo.
int available = aFile.available();

    // Asigne un almacenamiento intermedio que contenga los datos.
byte[] data = new byte[10240];

    // Lea todo el archivo, de 10 en 10 K.
for (int i = 0; i < available; i += 10240)
{
aFile.read(data);
}

    // Cierre el archivo.
aFile.close();
```

Además de los métodos de FileInputStream, la clase IFSFileInputStream proporciona al programa Java las opciones siguientes:

- Bloquear y desbloquear bytes de la corriente. En IFSKey encontrará más información.
- Especificar una modalidad de compartimiento al abrir el archivo. En Modalidades de compartimiento encontrará más información.

Clase IFSTextFileInputStream

La clase IFSTextFileInputStream representa una corriente de datos de tipo carácter leídos en un archivo. Los datos leídos del objeto IFSTextFileInputStream se proporcionan al programa Java en un objeto String Java, por lo que siempre es Unicode. Al abrir el archivo, el objeto IFSTextFileInputStream determina el CCSID de los datos del archivo. Si los datos están almacenados en una codificación distinta de Unicode, el objeto IFSTextFileInputStream convierte los datos de la codificación del archivo a Unicode antes de dárselos al programa Java. Si no es posible convertir los datos, se lanza una excepción UnsupportedEncodingException.

El ejemplo que sigue muestra cómo se utiliza la clase IFSTextFileInputStream:

```
        // Trabaje con /File en el sistema
        // mySystem.
    AS400 as400 = new AS400("mySystem");
    IFSTextFileInputStream file = new IFSTextFileInputStream(as400,
"/File");

        // Lea los cuatro primeros caracteres
        // del archivo.
    String s = file.read(4);

        // Visualice los caracteres leídos. Lea
        // los cuatro primeros caracteres del
        // archivo. De ser necesario, el objeto
        // IFSTextFileInputStream convierte los
        // datos a Unicode.
    System.out.println(s);

        // Cierre el archivo.
    file.close();
```

IFSFileOutputStream

La clase IFSFileOutputStream representa una corriente de salida para escribir datos en un archivo en el servidor. Al igual que en la clase IFSFile, en IFSFileOutputStream hay métodos que duplican los métodos de FileOutputStream del paquete java.io. IFSFileOutputStream dispone asimismo de métodos adicionales específicos para el servidor. La clase IFSFileOutputStream permite a un programa Java realizar estas tareas:

- Abrir un archivo para escritura. El archivo, si ya existe, se sustituye. Existen constructores que permiten especificar la modalidad de compartimiento de archivo y si se ha añadido el contenido de un archivo existente.
- Escribir bytes en la corriente.
- Comprometer en el disco los bytes que se escriben en la corriente.
- Bloquear o desbloquear bytes de la corriente.
- Cerrar el archivo.

Al igual que la clase FileOutputStream de java.io, esta clase permite a un programa Java escribir secuencialmente una corriente de bytes en el archivo.

El ejemplo que sigue muestra cómo se utiliza la clase IFSFileOutputStream.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Abra un objeto archivo que
    // represente el archivo.
IFSFileOutputStream aFile =
              new IFSFileOutputStream(sys,"/mydir1/mydir2/myfile");

    // Escriba en el archivo.
byte i = 123;
   aFile.write(i);

    // Cierre el archivo.
aFile.close();
```

Además de los métodos de FileOutputStream, IFSFileOutputStream proporciona al programa Java las opciones siguientes:

- Bloquear y desbloquear bytes de la corriente. En IFSKey encontrará más información.
- Especificar una modalidad de compartimiento al abrir el archivo. En Modalidades de compartimiento encontrará más información.

Clase IFSTextFileOutputStream

La clase IFSTextFileOutputStream representa una corriente de datos de tipo carácter que se están escribiendo en un archivo. Los datos proporcionados al objeto IFSTextFileOutputStream están en un objeto String Java, por lo que la entrada siempre es Unicode. Sin embargo, el objeto IFSTextFileOutputStream puede convertir los datos a otro CCSID a medida que se escriben en el archivo. El comportamiento por omisión es escribir caracteres Unicode en el archivo, pero el programa Java puede establecer el CCSID destino antes de que se abra el archivo. En este caso, el objeto IFSTextFileOutputStream convierte los caracteres de Unicode al CCSID especificado, antes de escribirlos en el archivo. Si no es posible convertir los datos, se lanza una excepción UnsupportedEncodingException.

El ejemplo que sigue muestra cómo se utiliza la clase IFSTextFileOutputStream:

```
        // Trabaje con /File en el sistema
        // mySystem.
    AS400 as400 = new AS400("mySystem");
    IFSTextFileOutputStream file = new IFSTextFileOutputStream(as400,
"/File");

        // Escriba una serie (String) en el archivo.
        // Debido a que no se ha especificado ningún
        // CCSID antes de escribir en el archivo,
        // se escribirán en él caracteres
        // Unicode. En el archivo se pondrá
        // una marca que indique que tiene
        // datos Unicode.
    file.write("Hola a todos");

        // Cierre el archivo.
    file.close();
```

IFSRandomAccessFile

La clase IFSRandomAccessFile representa un archivo existente en el servidor para la lectura y la escritura de datos. El programa Java puede leer y escribir datos de forma secuencial o aleatoria. Al igual que en la clase IFSFile, en IFSRandomAccessFile hay métodos que duplican los métodos de RandomAccessFile del paquete java.io. Además de estos métodos, IFSRandomAccessFile dispone de métodos adicionales específicos para el servidor iSeries o AS/400e. Mediante IFSRandomAccessFile, un programa Java puede llevar a cabo estas tareas:

- Abrir un archivo para acceso de lectura, escritura o lectura/escritura. Un programa Java puede especificar opcionalmente la modalidad de compartimiento de archivo y la opción de existencia.
- Leer datos en el desplazamiento actual del archivo.
- Escribir datos en el desplazamiento actual del archivo.
- Obtener o establecer el desplazamiento actual del archivo.
- Cerrar el archivo.

El ejemplo que sigue muestra cómo se utiliza la clase IFSRandomAccessFile para escribir cuatro bytes a intervalos de 1 K, en un archivo.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Abra un objeto archivo que represente
    // el archivo.
IFSRandomAccessFile aFile =
             new IFSRandomAccessFile(sys,"/mydir1/myfile", "rw");

    // Establezca los datos que han de escribirse.
byte i = 123;

    // Escriba en el archivo 10 veces a
    // intervalos de 1 K.
for (int j=0; j<10; j++)
{
    // Mueva el desplazamiento actual.
   aFile.seek(j * 1024);

    // Escriba en el archivo. El
    // desplazamiento actual avanza
    // el tamaño de la escritura.
   aFile.write(i);
}

    // Cierre el archivo.
aFile.close();
```

Además de los métodos de la clase RandomAccessFile de java.io, IFSRandomAccessFile proporciona al programa Java las opciones siguientes:

- Comprometer en disco los bytes escritos.
- Bloquear o desbloquear bytes del archivo.
- Bloquear y desbloquear bytes de la corriente. En IFSKey encontrará más información.
- Especificar una modalidad de compartimiento al abrir el archivo. En Modalidades de compartimiento encontrará más información.
- Especificar la opción de existencia al abrir un archivo. El programa Java puede elegir una de estas posibilidades:

- Si el archivo existe, abrirlo; si el archivo no existe, crearlo.
- Si el archivo existe, sustituirlo; si el archivo no existe, crearlo.
- Si el archivo existe, no realizar la apertura; si el archivo no existe, crearlo.
- Si el archivo existe, abrirlo; si el archivo no existe, no realizar la apertura.
- Si el archivo existe, sustituirlo; si el archivo no existe, no realizar la apertura.

IFSFileDialog

La clase IFSFileDialog permite recorrer el sistema de archivos y seleccionar un archivo. Esta clase utiliza la clase IFSFile para recorrer la lista de directorios y archivos del sistema de archivos integrado en el servidor iSeries o AS/400e. Los métodos de la clase permiten a un programa Java establecer el texto en los pulsadores del diálogo y establecer filtros. Observe que también está disponible una clase IFSFileDialog basada en Swing 1.1.

Puede establecer filtros mediante la clase FileFilter. Si el usuario selecciona un archivo en el diálogo, se puede utilizar el método getFileName() para obtener el nombre del archivo seleccionado. Se puede emplear el método getAbsolutePath() para obtener la vía de acceso y el nombre del archivo seleccionado.

El siguiente ejemplo muestra cómo se prepara un diálogo con dos filtros y se establece el texto en los pulsadores del diálogo.

```
        // Cree un objeto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

        // Cree un objeto diálogo estableciendo
        // el texto de la barra de título del
        // diálogo y el servidor que se debe recorrer.
     IFSFileDialog dialog = new IFSFileDialog(this, "Texto de barra de
título", sys);

        // Cree una lista de filtros y luego establezca
        // los filtros en el diálogo. Se utilizará
        // el primer filtro cuando el diálogo se
        // visualice por primera vez.
     FileFilter[] filterList = {new FileFilter("Todos los archivos (*.*)",
"*.*"),
                                new FileFilter("Archivos HTML (*.HTML",
"*.HTM")};

     dialog.setFileFilter(filterList, 0);

        // Establezca el texto en los botones
        // del diálogo.
     dialog.setOkButtonText("Abrir");
     dialog.setCancelButtonText("Cancelar");

        // Muestre el diálogo. Si el usuario
        // seleccionó un archivo pulsando el botón
        // Abrir, obtenga el archivo seleccionado
        // por el usuario y visualícelo.
     if (dialog.showDialog() == IFSFileDialog.OK)
        System.out.println(dialog.getAbsolutePath());
```

Clase IFSKey

El programa Java, si permite a otros programas acceder a un archivo simultáneamente, puede bloquear bytes en el archivo por un tiempo. Durante ese tiempo, el programa tiene el uso exclusivo de esa sección del archivo. Cuando un bloqueo se realiza satisfactoriamente, las clases del sistema de archivos integrado devuelven un objeto IFSKey. Este objeto se proporciona al método unlock() para indicar cuáles son los bytes que se han de desbloquear. Al cerrar el archivo, el sistema desbloquea todos los bloqueos que aún quedan en el archivo (el sistema realiza un desbloqueo de todos los bloqueos que el programa no desbloqueó).

El ejemplo que sigue muestra cómo se utiliza la clase IFSKey.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Abra una corriente de entrada.
    // Este constructor se abre con
    // share_all para que los demás programas
    // puedan abrir este archivo.
IFSFileInputStream aFile =
              new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");

    // Bloquee el primer Kilobyte del
    // archivo. Ahora ninguna otra instancia
    // puede leer estos bytes.
IFSKey key = aFile.lock(1024);

    // Lea el primer Kilobyte del archivo.
byte data[] = new byte[1024];
aFile.read(data);

    // Desbloquee los bytes del archivo.
aFile.unlock(key);

    // Cierre el archivo.
aFile.close();
```

Modalidad de compartimiento de archivo

El programa Java puede especificar una modalidad de compartimiento al abrir un archivo. Puede ser que el programa permita a los demás programas abrir el archivo simultáneamente o bien que tenga un acceso exclusivo al archivo.

El ejemplo que sigue muestra cómo se especifica una modalidad de compartimiento de archivo:

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Abra un objeto archivo que
    // represente el archivo. Debido a que
    // este programa especifica no compartir,
    // todos los demás intentos de apertura fallan
    // mientras no se cierre esta instancia.
IFSFileOutputStream aFile =
              new IFSFileOutputStream(sys,
                                      "/mydir1/mydir2/myfile",
                                      IFSFileOutputStream.SHARE_NONE,
                                      false);

    // ...Realice operaciones en
    // el archivo.

    // Cierre el archivo. Ahora son satisfactorias
    // las demás operaciones de apertura.
aFile.close();
```

JavaApplicationCall

La clase JavaApplicationCall proporciona la posibilidad de que el cliente utilice la máquina virtual Java del servidor para ejecutar un programa Java que reside en el servidor.

Después de establecer una conexión con el servidor desde el cliente, la clase JavaApplicationCall le permite realizar estas tareas de configuración:

1. Establecer la variable de entorno CLASSPATH en el servidor con el método setClassPath()
2. Definir los parámetros del programa con el método setParameters()
3. Ejecutar el programa con run()
4. Enviar una entrada desde el cliente al programa Java. El programa Java lee la entrada por medio de una entrada estándar que se establece con el método sendStandardInString(). La salida estándar y la salida de error estándar se pueden redirigir desde el programa Java al cliente por medio de getStandardOutString() y getStandardErrorString()

JavaApplicationCall es una clase a la que se llama desde el programa Java. Sin embargo, IBM Toolbox para Java también proporciona programas de utilidad para llamar a programas Java que residen en el servidor. Estos programas de utilidad son programas Java completos que se pueden ejecutar desde la estación de trabajo. Puede encontrar más información en la clase RunJavaApplication.

Ejemplo

Este ejemplo muestra cómo se ejecuta un programa en el servidor desde el cliente cuya salida es "¡Hola a todos!".

JDBC

JDBC(TM) es una interfaz de programas de aplicación (API) incluida en la plataforma Java que permite a los programas Java conectar con una gran variedad de bases de datos.

El controlador JDBC de IBM Toolbox para Java permite utilizar interfaces API JDBC para emitir sentencias SQL (Structured Query Language) a las bases de datos del servidor y procesar los resultados procedentes de las bases de datos del servidor. »También puede emplear el controlador JDBC de IBM Developer Kit para Java, denominado el controlador JDBC 'nativo':

- Utilice el controlador JDBC de la Caja de Herramientas cuando el programa Java esté en un sistema y los archivos de base de datos se encuentren en otro sistema, como en un entorno de cliente/servidor.
- Utilice el controlador JDBC nativo cuando tanto el programa Java como los archivos de base de datos estén en el mismo sistema iSeries.

Para obtener información sobre las mejoras que están en curso, consulte las novedades de V5R2 y las mejoras efectuadas en el soporte JDBC de Toolbox para Java.«

Diferentes versiones de JDBC

Existen distintas versiones de la API JDBC; el controlador JDBC de IBM Toolbox para Java da soporte a las versiones siguientes:

- La API JDBC 1.2 (el paquete java.sql) se incluye en la API de núcleo de la plataforma Java 1.1 y JDK 1.1.
- La API de núcleo de JDBC 2.1 (el paquete java.sql) se incluye tanto en la plataforma Java 2, Standard Edition (J2SE) como en la plataforma Java 2, Enterprise Edition (J2EE).
- La API JDBC 2.0 Optional Package (el paquete javax.sql) se incluye en J2EE y está disponible como producto de Sun que puede bajarse de forma independiente. Estas extensiones antiguamente se denominaban API de extensiones estándar de JDBC 2.0.
- »La API JDBC 3.0 (los paquetes java.sql y javax.sql) se incluye en J2SE, Versión 1.4.«

Interfaces soportadas

La tabla siguiente muestra las interfaces JDBC soportadas y la API necesaria para utilizarlas:

Interfaz JDBC soportada	API necesaria
Blob proporciona acceso a los objetos binarios de gran tamaño (BLOB).	JDBC 2.1 de núcleo
CallableStatement ejecuta procedimientos almacenados de SQL.	JDK 1.1
Clob proporciona acceso a los objetos de tipo carácter de gran tamaño (CLOB).	JDBC 2.1 de núcleo
Connection representa una conexión con una base de datos específica.	JDK 1.1
»ConnectionPool representa una agrupación de objetos Connection.	JDBC 2.0 Optional Package«
ConnectionPoolDataSource representa una fábrica de objetos AS400JDBCPooledConnection reunidos en una agrupación.	JDBC 2.0 Optional Package
DatabaseMetaData proporciona información acerca de la base de datos como conjunto.	JDK 1.1
DataSource representa una fábrica de conexiones de base de datos.	JDBC 2.0 Optional Package
Driver crea la conexión y devuelve información acerca de la versión del controlador.	JDK 1.1
»ParameterMetaData ofrece la posibilidad de obtener información acerca de los tipos y las propiedades de los parámetros de un objeto PreparedStatement.	API JDBC 3.0«

PreparedStatement ejecuta sentencias SQL compiladas	JDK 1.1
ResultSet proporciona acceso a una tabla de datos que se genera mediante la ejecución de una consulta SQL o el método catalog de DatabaseMetaData.	JDK 1.1
ResultSetMetaData proporciona información sobre un conjunto de resultados (ResultSet) específico.	JDK 1.1
RowSet es un conjunto de filas conectado que encapsula un conjunto de resultados (ResultSet).	JDBC 2.0 Optional Package
»Savepoint proporciona un control más específico en las transacciones.	API JDBC 3.0«
Statement ejecuta sentencias SQL y obtiene los resultados.	JDK 1.1
XAConnection es una conexión de base de datos que participa en transacciones XA globales.	JDBC 2.0 Optional Package
XAResource es un gestor de recursos destinado a su uso en las transacciones XA.	JDBC 2.0 Optional Package

Hemos incluido una tabla en la que figuran las propiedades de JDBC para facilitar su consulta.

Ejemplos

Los ejemplos siguientes muestran formas de utilizar el controlador JDBC de IBM Toolbox para Java.

- Cómo se utiliza el controlador JDBC para crear y llenar con datos un a tabla
- Cómo se utiliza el controlador JDBC para consultar una tabla y enviar su contenido a la salida

Mejoras efectuadas en el soporte JDBC de Toolbox para Java

Entre las funciones JDBC de OS/400 Versión 5 Release 2 mejoradas se encuentran:

- Eliminación de la restricción FOR UPDATE
- Cambio en el truncamiento de datos
- Obtener y modificar columnas y parámetros por nombre
- Recuperar claves generadas automáticamente
- Rendimiento mejorado al ejecutar sentencias SQL insert en un proceso por lotes
- Soporte mejorado para ResultSet.getRow()
- Soporte mejorado para utilizar mayúsculas y minúsculas combinadas en nombres de columna
- Especificar la posibilidad de retención de los objetos Statement, CallableStatement y PreparedStatement
- Soporte de aislamiento de transacción mejorado

Eliminación de la restricción FOR UPDATE

Ya no es necesario que especifique FOR UPDATE en las sentencias SELECT para garantizar un cursor actualizable. Al conectarse a la versión de OS/400 V5R1 y posteriores, Toolbox para Java respeta todas las concurrencias que el usuario pasa al crear sentencias. El valor por omisión sigue siendo un cursor de sólo lectura si no se especifica ninguna concurrencia.

El **truncamiento de datos lanza excepciones únicamente cuando se escriben datos de tipo carácter truncados en la base de datos**

Las reglas del truncamiento de datos de Toolbox para Java son las mismas que las del controlador JDBC de IBM Developer Kit para Java. Para obtener más información, consulte las propiedades de JDBC de IBM Toolbox para Java.

Obtener y modificar columnas y parámetros por nombre

Varios métodos nuevos permiten obtener y actualizar información por nombre de columna en ResultSet y obtener y establecer información por nombre de parámetro en CallableStatement. Por ejemplo, en ResultSet, donde antes utilizaba lo siguiente:

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

Ahora puede emplear:

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

Tenga en cuenta que al acceder a parámetros por su índice se obtiene un mejor rendimiento que al acceder a ellos por su nombre. También puede especificar nombres de parámetro para establecerlos en CallableStatement. Donde podía haber utilizado lo siguiente en CallableStatement:

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

Ahora puede emplear:

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

Para emplear estos métodos nuevos, necesita JDBC 3.0 o posterior y la plataforma Java 2, versión 1.4

(Standard o Enterprise Edition).

Recuperar claves generadas automáticamente

El método getGeneratedKeys() de AS400JDBCStatement recupera las claves generadas automáticamente que se han creado como consecuencia de la ejecución de ese objeto Statement. Cuando el objeto Statement no genera ninguna clave, se devuelve un objeto ResultSet vacío. Actualmente el servidor soporta únicamente la devolución de una clave generada automáticamente (la clave de la última fila insertada). En el ejemplo siguiente se muestra cómo insertar un valor en una tabla y, a continuación, obtener la clave generada automáticamente:

```
     Statement s = statement.executeQuery
          ("INSERT INTO MYSCHOOL/MYSTUDENTS (FIRSTNAME) VALUES ('JOHN'");
     ResultSet rs = s.getGeneratedKeys();
          // Actualmente el servidor iSeries soporta únicamente la
devolución de una
          // clave generada automáticamente (la clave de la última fila
insertada).
            rs.next ();
     String autoGeneratedKey = rs.getString(1);
          // Utilice la clave generada automáticamente, por ejemplo, como
clave primaria en otra tabla.
```

Para recuperar las claves generadas automáticamente, necesita JDBC 3.0 o posterior y la plataforma Java 2, versión 1.4 (Standard o Enterprise Edition). Para recuperar las claves generadas automáticamente también es necesario conectarse a una versión V5R2 o posterior de OS/400.

Rendimiento mejorado al ejecutar sentencias SQL insert en un proceso por lotes

El rendimiento que se obtiene al ejecutar sentencias SQL insert en un proceso por lotes ha mejorado. Puede ejecutar sentencias SQL en un proceso por lotes empleando los distintos métodos addBatch() disponibles en AS400JDBCStatement, AS400JDBCPreparedStatement y AS400JDBCCallableStatement. El soporte de proceso por lotes mejorado sólo afecta a las peticiones de inserción. Por ejemplo, al utilizar el soporte de proceso por lotes para procesar varias inserciones sólo se pasa una petición al servidor. No obstante, al utilizar el soporte de proceso por lotes para procesar una inserción, una actualización y una supresión, se envía cada petición de forma individual.

Para emplear el soporte de proceso por lotes, necesita JDBC 2.0 o posterior y la plataforma Java 2, versión 1.2 (Standard o Enterprise Edition).

Soporte mejorado para ResultSet.getRow()

Antes, el controlador JDBC de IBM Toolbox para Java tenía restricciones en el soporte para el método getRow() en ResultSet. En concreto, al utilizar ResultSet.last(), ResultSet.afterLast() y ResultSet.absolute() con un valor negativo, el número de fila actual no estaba disponible. Las restricciones anteriores se han eliminado, con lo que este método es ahora totalmente funcional.

Utilización de mayúsculas y minúsculas combinadas en nombres de columna

Los métodos de Toolbox para Java deben emparejar los nombres de columna proporcionados por el usuario o los nombres de columna proporcionados por la aplicación con los nombres que se encuentran en la tabla de base de datos. En cualquier caso, cuando un nombre de columna no está delimitado por comillas, Toolbox para Java cambia el nombre a mayúsculas antes de compararlo con los nombres del servidor. Si el nombre de columna está delimitado por comillas, debe coincidir exactamente con el nombre del servidor; de lo contrario, Toolbox para Java lanza una excepción.

Especificar la posibilidad de retención de los objetos Statement, CallableStatement y PreparedStatement creados

Los nuevos métodos de AS400JDBCConnection permiten especificar la posibilidad de retención de los objetos Statement, CallableStatement y PreparedStatement que se crean. La posibilidad de retención determina si los cursores se mantienen abiertos o cerrados al comprometer la transacción. Ahora puede hacer que una sentencia tenga una posibilidad de retención distinta a la de su objeto de conexión. Asimismo, los objetos de conexión pueden tener varios objetos de sentencia abiertos, cada uno de ellos con una posibilidad de retención distinta especificada. Al llamar a commit, cada sentencia se maneja según la posibilidad de retención especificada para esa sentencia.

La posibilidad de retención se obtiene a partir del siguiente orden de prioridad:

1. La posibilidad de retención especificada en la creación de la sentencia mediante los métodos de la clase Connection createStatement(), prepareCall() o prepareStatement().
2. La posibilidad de retención especificada mediante Connection.setHoldability(int).
3. La posibilidad de retención especificada por la propiedad cursor hold de JDBC de Toolbox para Java (cuando no se utilizan los métodos de 1. o 2.).

Para emplear estos métodos, necesita JDBC 3.0 o posterior y la plataforma Java 2, versión 1.4 (Standard o Enterprise Edition). Asimismo, los servidores que ejecutan una versión V5R1 o anterior de OS/400 sólo pueden utilizar la posibilidad de retención especificada por la propiedad cursor hold de JDBC.

Soporte de aislamiento de transacción mejorado

El controlador JDBC de IBM Toolbox para Java ahora ofrece soporte para conmutar al nivel de aislamiento de transacción *NONE tras efectuarse una conexión. Antes de la versión V5R2, el controlador JDBC de Toolbox para Java lanzaba una excepción al conmutar a *NONE tras efectuar una conexión.

IBM Toolbox para Java: propiedades de JDBC

Pueden especificarse muchas propiedades al conectar con una base de datos de servidor utilizando JDBC. Todas las propiedades son opcionales y pueden especificarse como parte del URL o en un objeto java.util.Properties. Si se establece una propiedad tanto en el URL como en un objeto Properties, se empleará el valor del URL.

Nota: la lista siguiente no incluye las propiedades de origen de datos.

Las tablas siguientes muestran las distintas propiedades de conexión que este controlador reconoce. Algunas de estas propiedades afectan al rendimiento y otras son atributos de trabajo servidor. Las tablas organizan las propiedades en las categorías siguientes:

Propiedades generales

Las propiedades generales son atributos del sistema que especifican el usuario, la contraseña y si se necesita una solicitud para conectarse al servidor.

Propiedad general	Descripción	Obligatoria	Opciones	Valor por omisión
"password"	Especifica la contraseña para conectarse al servidor. Si no se especifica ninguna, se le solicitará al usuario, salvo que se haya establecido la propiedad "prompt" en "false", en cuyo caso el intento de conexión fallará.	no	contraseña del servidor	(se solicitará al usuario)
"prompt"	Especifica si debe presentarse al usuario una solicitud en el caso de que sea necesario un nombre de usuario o una contraseña para conectarse al servidor. Si no puede establecerse una conexión sin solicitar al usuario y esta propiedad se establece en "false", el intento de conexión fallará.	no	"true" "false"	"true"
"user"	Especifica el nombre de usuario para conectarse al servidor. Si no se especifica ninguno, se le solicitará al usuario, salvo que se haya establecido la propiedad "prompt" en "false", en cuyo caso el intento de conexión fallará.	no	usuario del servidor	(se solicitará al usuario)

Propiedades de servidor

Las propiedades de servidor especifican atributos que rigen las transacciones, las bibliotecas y las bases de datos.

Propiedad de servidor	Descripción	Obligatoria	Opciones	Valor por omisión

"cursor hold"	Especifica si se debe retener el cursor entre transacciones. Si esta propiedad se establece en "true", los cursores no se cierran cuando se compromete o retrotrae una transacción. Todos los recursos obtenidos durante la unidad de trabajo se retienen, pero los bloqueos sobre filas y objetos específicos obtenidos implícitamente durante la unidad de trabajo se liberan.	no	"true" "false"	"true"
»"cursor sensivity"	Especifica la sensibilidad de cursor solicitada desde la base de datos. El comportamiento depende del resultSetType: ● ResultSet.TYPE_FORWARD_ONLY o ResultSet.TYPE_SCROLL_SENSITIVE indica que el valor de esta propiedad controla la sensibilidad que el programa Java solicita a la base de datos. ● ResultSet.TYPE_SCROLL_INSENSITIVE hace que se ignore esta propiedad. Esta propiedad se ignora en las conexiones con sistemas que ejecutan V5R1 y versiones anteriores de OS/400.	no	"asensitive" "insensitive" "sensitive"	"asensitive"«
»"database name"	Especifica la base de datos que debe utilizarse para la conexión, incluidas las almacenadas en una agrupación de almacenamiento auxiliar independiente. Esta propiedad sólo es válida al conectarse a una versión V5R2 o posterior de OS/400. Cuando especifique un nombre de base de datos, el nombre debe existir en el directorio de bases de datos relacionales del servidor. Los criterios siguientes determinan a qué base de datos se accede: ● Cuando se utiliza esta propiedad para especificar una base de datos, se utiliza la base de datos especificada. Si la base de datos no existe, la conexión falla. ● Cuando se utiliza esta propiedad para especificar *SYSBAS como nombre de base de datos, se utiliza la base de datos por omisión del sistema. ● Cuando se omite esta propiedad, se utiliza el nombre de base de datos especificado en la descripción de trabajo del perfil de usuario. Cuando la descripción de trabajo no especifica ningún nombre de base de datos, se utiliza la base de datos por omisión del sistema.	no	Nombre de base de datos "*SYSBAS"	Se utiliza el nombre de base de datos especificado en la descripción de trabajo del perfil de usuario. Cuando la descripción de trabajo no especifica ningún nombre de base de datos, se utiliza la base de datos por omisión del sistema.«
"libraries"	Especifica una o varias bibliotecas que desea añadir en o sustituir por la lista de bibliotecas del trabajo del servidor, y opcionalmente establece la biblioteca por omisión (esquema por omisión). **Lista de bibliotecas** El servidor utiliza las bibliotecas especificadas para resolver los nombres de procedimientos almacenados no calificados y los procedimientos almacenados las utilizan para resolver nombres no calificados. Para especificar varias bibliotecas, utilice comas o espacios para separar las distintas entradas. Puede utilizar *LIBL para indicar la lista de bibliotecas actual del trabajo del servidor: Cuando la primera entrada es *LIBL, las bibliotecas especificadas se añaden a la lista de bibliotecas actual del trabajo del servidor. Si no indica *LIBL, las bibliotecas	no	Lista de bibliotecas del servidor, separadas por comas o espacios	"*LIBL"

	especificadas sustituyen la lista de bibliotecas actual del trabajo del servidor. **Esquema por omisión** El servidor utiliza el esquema por omisión para resolver los nombres no calificados en las sentencias SQL. Por ejemplo, en la sentencia "SELECT * FROM MITABLA", el servidor sólo buscará MITABLA en el esquema por omisión. Puede especificar el esquema por omisión en el URL de conexión. Si no especifica el esquema por omisión en el URL de conexión, son aplicables las siguientes condiciones, en función de si se utilizan denominaciones SQL o denominaciones del sistema. ● Denominaciones SQL Si no especifica el esquema por omisión en el URL de conexión: ❍ La primera entrada (si no es *LIBL) pasa a aser el esquema por omisión ❍ Si la primera entrada es *LIBL, la segunda entrada pasa a ser el esquema por omisión ❍ Si no establece esta propiedad o si solamente contiene *LIBL, el perfil de usuario pasa a ser el esquema por omisión ● Denominaciones del sistema Si no especifica el esquema por omisión en el URL de conexión: ❍ No se establece ningún esquema por omisión y el servidor utiliza las bibliotecas especificadas para buscar en ellas los nombres no calificados ❍ Si no establece esta propiedad o si solamente contiene *LIBL, el servidor utiliza la lista de bibliotecas actual del trabajo del servidor para buscar en ellas los nombres no calificados			
"transaction isolation"	Especifica el aislamiento de transacción por omisión.	no	"none" "read uncommitted" "read committed" "repeatable read" "serializable"	"read uncommitted"

Propiedades de formato

Las propiedades de formato especifican los formatos de fecha y hora, los separadores de fecha y decimales y los convenios de denominación de tablas empleados en las sentencias SQL.

Propiedad de formato	Descripción	Obligatoria	Opciones	Valor por omisión

"date format"	Especifica el formato de fecha utilizado en los literales de fecha dentro de las sentencias SQL.	no	"mdy" "dmy" "ymd" "usa" "iso" "eur" "jis" "julian"	(trabajo servidor)
"date separator"	Especifica el separador de fecha utilizado en los literales de fecha dentro de las sentencias SQL. Esta propiedad no surte ningún efecto salvo que la propiedad "date format" se haya establecido en "julian", "mdy", "dmy" o "ymd".	no	"/" (barra inclinada) "-" (guión) "." (punto) "," (coma) "b" (espacio)	(trabajo servidor)
"decimal separator"	Especifica el separador decimal utilizado en los literales numéricos dentro de las sentencias SQL.	no	"." (punto) "," (coma)	(trabajo servidor)
"naming"	Especifica el convenio de denominación utilizado al hacer referencia a las tablas.	no	"sql" (como en `esquema.tabla`) "system" (como en `esquema/tabla`)	"sql"
"time format"	Especifica el formato de hora utilizado en los literales de hora dentro de las sentencias SQL.	no	"hms" "usa" "iso" "eur" "jis"	(trabajo servidor)
"time separator"	Especifica el separador de hora utilizado en los literales de hora dentro de las sentencias SQL. Esta propiedad no surte ningún efecto salvo que la propiedad "time format" se haya establecido en "hms".	no	":" (dos puntos) "." (punto) "," (coma) "b" (espacio)	(trabajo servidor)

Propiedades de rendimiento

Las propiedades de rendimiento son atributos que incluyen el almacenamiento en antememoria, la conversión de datos, la compresión de datos y la prebúsqueda que afectan al rendimiento.

Propiedad de rendimiento	Descripción	Obligatoria	Opciones	Valor por omisión

"big decimal"	Especifica si se utiliza un objeto java.math.BigDecimal intermedio para conversiones de decimal empaquetado y con zona. Si esta propiedad se establece en "true", se utiliza un objeto java.math.BigDecimal intermedio para conversiones de decimal empaquetado y con zona tal como describe la especificación de JDBC. Si esta propiedad se establece en "false", no se utiliza ningún objeto intermedio para conversiones de decimal empaquetado y con zona. En su lugar, estos valores se convierten directamente desde valores dobles Java y en valores dobles Java. Estas conversiones serán más rápidas pero puede que no sigan todas las reglas de conversión y truncamiento de datos documentadas en la especificación de JDBC.	no	"true" "false"	"true"
"block criteria"	Especifica los criterios para recuperar datos del servidor en bloques de registros. Si se especifica un valor distinto de cero para esta propiedad se reducirá la frecuencia de comunicación con el servidor, lo que aumentará el rendimiento. Compruebe que la función de bloques de registros esté desactivada si se va a utilizar el cursor para posteriores operaciones de actualización (UPDATE); de lo contrario, la fila que se actualice no será necesariamente la fila actual.	no	"0" (sin bloques de registros) "1" (con bloques si se especifica FOR FETCH ONLY) "2" (con bloques salvo que se especifique FOR UPDATE)	"2"
"block size"	Especifica el tamaño de bloque (en kilobytes) que debe recuperarse del servidor y colocarse en la antememoria del cliente. Esta propiedad no surte ningún efecto salvo que la propiedad "block criteria" se haya establecido en un valor distinto de cero. Si se especifican grandes tamaños de bloque se reducirá la frecuencia de comunicación con el servidor, lo que aumentará el rendimiento.	no	"0" "8" "16" "32" "64" "128" "256" "512"	"32"
"data compression"	Especifica si los datos del conjunto de resultados se comprimen. Si esta propiedad se establece en "true", los datos del conjunto de resultados se comprimen. Si esta propiedad se establece en "false", los datos del conjunto de resultados no se comprimen. La compresión de datos puede mejorar el rendimiento al recuperar conjuntos de resultados de gran tamaño.	no	"true" "false"	"true"

"extended dynamic"	Especifica si debe utilizarse soporte dinámico ampliado. El soporte dinámico ampliado proporciona un mecanismo para poner en antememoria en el servidor las sentencias SQL dinámicas. »La primera vez que se ejecuta una sentencia SQL concreta, se almacena en un paquete SQL en el servidor. Si el paquete no existe, se crea automáticamente. En posteriores preparaciones de la misma sentencia SQL, el servidor puede saltarse una parte notable del proceso utilizando la información almacenada en el paquete SQL.« Si esta propiedad se establece en "true", debe establecerse un nombre de paquete con la propiedad "package".	no	"true" "false"	"false"
"lazy close"	Especifica si hay que diferir el cierre de los cursores hasta las peticiones ulteriores. Esto aumentará el rendimiento global al reducir el número total de peticiones.	no	"true" "false"	"false"
"lob threshold"	Especifica el tamaño máximo de LOB (objeto de gran tamaño) en bytes que puede recuperarse como parte de un conjunto de resultados. Los LOB cuyo tamaño sobrepase este umbral se recuperarán en fragmentos, utilizando una comunicación adicional con el servidor. Los umbrales de LOB de mayor tamaño reducirán la frecuencia de la comunicación con el servidor, pero bajarán más datos de LOB, aunque dichos datos no se utilicen. Los umbrales de LOB de menor tamaño pueden incrementar la frecuencia de la comunicación con el servidor, pero únicamente bajarán los datos de LOB según se necesiten.	no	"0" - "16777216"	"0"
"package"	Especifica el nombre base del paquete SQL. »Observe que sólo se utilizan los siete primeros caracteres para generar el nombre del paquete SQL en el servidor.« Esta propiedad no surte ningún efecto salvo que la propiedad "extended dynamic" se haya establecido en "true". Además, esta propiedad debe establecerse si se ha establecido la propiedad "extended dynamic" en "true".	no	Paquete SQL	""
"package add"	»Especifica si deben añadirse las nuevas sentencias preparadas al paquete SQL especificado en la propiedad "package". Esta propiedad no surte ningún efecto salvo que la propiedad "extended dynamic" se haya establecido en "true".«	no	"true" "false"	"true"

"package cache"	»Especifica si debe almacenarse en antememoria un subconjunto de la información del paquete SQL en la memoria del cliente. Al almacenar paquetes SQL en la antememoria local, se reduce la cantidad de comunicación con el servidor para las preparaciones y descripciones. Esta propiedad no surte ningún efecto salvo que la propiedad "extended dynamic" se haya establecido en "true".«	no	"true" "false"	"false"
"package criteria"	Especifica el tipo de sentencias SQL que deben almacenarse en el paquete SQL. Esto puede resultar de utilidad para mejorar el rendimiento de las condiciones de unión complejas. Esta propiedad no surte ningún efecto salvo que la propiedad "extended dynamic" se haya establecido en "true".	no	"default" (almacenar únicamente las sentencias SQL con marcadores de parámetros en el paquete) »"select" (almacenar todas las sentencias SQL SELECT en el paquete)«	"default"
"package error"	Especifica la acción que debe efectuarse cuando se producen errores de paquete SQL. Cuando se produce un error de paquete SQL, el controlador puede lanzar una SQLException o enviar un aviso al objeto Connection, según el valor de esta propiedad. Esta propiedad no surte ningún efecto salvo que la propiedad "extended dynamic" se haya establecido en "true".	no	"exception" "warning" "none"	"warning"
"package library"	Especifica la biblioteca para el paquete SQL. Esta propiedad no surte ningún efecto salvo que la propiedad "extended dynamic" se haya establecido en "true".	no	Biblioteca para el paquete SQL	"QGPL"
"prefetch"	Especifica si deben prebuscarse datos al ejecutar una sentencia SELECT. Esto incrementará el rendimiento al acceder a las filas iniciales del conjunto de resultados (ResultSet).	no	"true" "false"	"true"

Propiedades de ordenación

Las propiedades de ordenación especifican cómo lleva a cabo el servidor las operaciones de almacenar y ordenar.

Propiedad de ordenación	Descripción	Obligatoria	Opciones	Valor por omisión

"sort"	Especifica cómo ordena el servidor los registros antes de enviarlos al cliente.	no	"hex" (la ordenación se basa en los valores hexadecimales) "job" (la ordenación se basa en el valor correspondiente al trabajo servidor) "language" (la ordenación se basa en el idioma establecido en la propiedad "sort language") "table" (la ordenación se basa en la tabla de secuencia de ordenación establecida en la propiedad "sort table")	"job"
"sort language"	Especifica un ID de idioma de tres caracteres que debe utilizarse para la selección de una secuencia de ordenación. Esta propiedad no surte ningún efecto salvo que la propiedad "sort" se haya establecido en "language".	no	ID de idioma	ENU
"sort table"	Especifica la biblioteca y el nombre de archivo de una tabla de secuencia de ordenación almacenada en el servidor. Esta propiedad no surte ningún efecto salvo que la propiedad "sort" se haya establecido en "table".	no	Nombre calificado de tabla de ordenación	""
"sort weight"	Especifica cómo trata el servidor las mayúsculas/minúsculas al ordenar registros. Esta propiedad no surte ningún efecto salvo que la propiedad "sort" se haya establecido en "language".	no	"shared" (los caracteres en mayúsculas y en minúsculas se ordenan como el mismo carácter) "unique" (los caracteres en mayúsculas y en minúsculas se ordenan como caracteres distintos)	"shared"

Otras propiedades

Otras propiedades son las que no pueden incluirse fácilmente en categorías. Estas propiedades determinan qué controlador JDBC se utiliza y especifican opciones relacionadas con el nivel de acceso a la base de datos, el tipo de serie bidireccional, el truncamiento de datos, etc.

Otra propiedad	Descripción	Obligatoria	Opciones	Valor por omisión
"access"	Especifica el nivel de acceso a base de datos para la conexión.	no	"all" (todas las sentencias SQL permitidas) "read call" (todas las sentencias SELECT y CALL permitidas) "read only" (sólo las sentencias SELECT)	"all"

»"behavior override"	Especifica los comportamientos del controlador JDBC de IBM Toolbox para Java que deben alterarse temporalmente. Puede cambiar varios comportamientos mediante la adición de constantes y el paso de dicha suma a esta propiedad. Asegúrese de que la aplicación maneja correctamente los comportamientos alterados.	no	"" (no alterar temporalmente ningún comportamiento) "1" (no lanzar ninguna excepción pero devolver nulo para el conjunto de resultados si Statement.executeQuery() o PreparedStatement.executeQuery() no devuelven un conjunto de resultados)	""«
"bidirectional string type"	Especifica el tipo de serie de salida de los datos bidireccionales. En BidiStringType encontrará más información.	no	"" (utilizar el CCSID para determinar el tipo de serie bidireccional) "0" (el tipo de serie por omisión para los datos no bidireccionales (LTR)) "4" "5" "6" "7" "8" "9" "10" "11"	""
"data truncation"	»Especifica si el truncamiento de **datos de tipo carácter** genera avisos y excepciones. Cuando esta propiedad es "true", se aplican las reglas siguientes: ● Al escribir datos de tipo carácter truncados en la base de datos se lanza una excepción. ● Al utilizar datos de tipo carácter truncados en una consulta se envía un aviso. Cuando esta propiedad es "false", al escribir datos truncados en la base de datos o utilizar esos datos en una consulta no se generan ni excepciones ni avisos. El valor por omisión es "true". Esta propiedad no afecta a los **datos numéricos**. Al escribir datos numéricos truncados en la base de datos siempre se lanza un error y al utilizar datos numéricos truncados en una consulta siempre se envía un aviso.«	no	"true" "false"	"true"

"driver"	Especifica la implementación del controlador JDBC. El controlador JDBC de IBM Toolbox para Java puede utilizar distintas implementaciones del controlador JDBC en función del entorno. Si el entorno es una máquina virtual Java de iSeries situada en el mismo servidor que la base de datos a la que se conecta el programa, puede utilizarse el controlador JDBC de IBM Developer Kit para Java nativo. En cualquier otro entorno, se utiliza el controlador JDBC de IBM Toolbox para Java. Esta propiedad no surte ningún efecto si se ha establecido la propiedad "secondary URL".	no	"toolbox" (se utilizará únicamente el controlador JDBC de IBM Toolbox para Java) "native" (se utilizará el controlador JDBC de IBM Developer Kit para Java si se ejecuta en el servidor; de lo contrario, se utilizará el controlador JDBC de Toolbox para Java)	"toolbox"
"errors"	Especifica la cantidad de detalle que debe devolverse en el mensaje para los errores que se producen en el servidor.	no	"basic" "full"	"basic"
"extended metadata"	Especifica si el controlador debe solicitar metadatos ampliados procedentes del servidor. Al establecer esta propiedad en true se aumenta la precisión de la información devuelta por los siguientes métodos de ResultSetMetaData: • getColumnLabel(int) • isReadOnly(int) • isSearchable(int) • isWriteable(int) Asimismo, al establecer esta propiedad en true se habilita el soporte para el método ResultSetMetaData.getSchemaName(int). Al establecer esta propiedad en true puede reducirse el rendimiento ya que es preciso recuperar más información del servidor. Deje la propiedad establecida en el valor por omisión (false) salvo que necesite información más específica de los métodos listados. Por ejemplo, cuando esta propiedad está inactiva (false), ResultSetMetaData.isSearchable(int) siempre devuelve "true" ya que el controlador no tiene suficiente información procedente del servidor para hacer un juicio. Al activar esta propiedad (true), se fuerza al controlador a obtener los datos correctos del servidor. Únicamente puede emplear metadatos ampliados al conectarse a un servidor que ejecute OS/400 V5R2 o posterior.	no	"true" "false"	"false"

"full open"	Especifica si el servidor abre por completo un archivo para cada consulta. Por omisión, el servidor optimiza las peticiones abiertas. Esta optimización mejora el rendimiento pero puede fallar si hay un supervisor de base de datos activo cuando se ejecuta una consulta más de una vez. Establezca la propiedad en true únicamente si se emiten consultas idénticas cuando hay supervisores activos.	no	"true" "false"	"false"
"nombre de archivo de claves"	Especifica el nombre de la clase de archivo de claves utilizada para las conexiones SSL con el servidor. Esta propiedad no surte ningún efecto salvo que "secure" se haya establecido en true y que se haya establecido una contraseña de archivo de claves mediante la propiedad "key ring password".	no	"nombre de archivo de claves"	""
"contraseña de archivo de claves"	Especifica la contraseña para la clase de archivo de claves utilizada para las comunicaciones SSL con el servidor. Esta propiedad no surte ningún efecto salvo que "secure" se haya establecido en true y que se haya establecido un nombre de archivo de claves mediante la propiedad "key ring name".	no	"contraseña de archivo de claves"	""
"proxy server"	Especifica el nombre de sistema principal y el número de puerto de la máquina de número medio de niveles en la que se está ejecutando el servidor proxy. El formato es nombre_sistema_principal[:puerto], siendo el puerto opcional. Si no se establece esta propiedad, el nombre de sistema principal y el número de puerto se recuperan de la propiedad *com.ibm.as400.access.AS400.proxyServer*. El puerto por omisión es *3470* (si la conexión utiliza SSL, el puerto por omisión es *3471*). El servidor proxy debe estar en ejecución en la máquina de número medio de niveles. El nombre de la máquina de número medio de niveles se omite en un entorno de dos niveles.	no	Nombre de sistema principal y número de puerto de servidor proxy	(el valor de la propiedad proxyServer, o none si no está establecida)
"remarks"	Especifica el origen del texto para las columnas REMARKS de los objetos ResultSet devueltos por los métodos de DatabaseMetaData.	no	"sql" (comentario de objeto SQL) "system" (descripción de objeto OS/400)	"system"

"save password when serialized"	Especifica si debe guardarse la contraseña localmente con el resto de las propiedades, al serializar este objeto de origen de datos. Si se guarda la contraseña, la aplicación debe proteger la forma serializada del objeto, ya que el objeto contiene toda la información necesaria para conectarse al servidor. Antes de establecer esta propiedad en "true,", considere los riesgos de seguridad que puede ocasionar si se guarda la contraseña con el resto de propiedades.	no	"true" "false"	"false"
"secondary URL"	Especifica el URL que se utilizará para una conexión en el objeto DriverManager de número medio de niveles en un entorno de varios niveles, si es distinto del que ya se ha especificado. Esta propiedad permite utilizar este controlador para conectar con bases de datos distintas del servidor iSeries o AS/400e. Emplee una barra inclinada invertida como carácter de escape antes de los caracteres de barra inclinada invertida y punto y coma en el URL.	no	URL JDBC	(URL JDBC actual)
"secure"	Especifica si para comunicarse con el servidor se utiliza una conexión SSL (capa de sockets segura). Las conexiones SSL sólo están disponibles al conectar con servidores con la versión V4R4 o posterior.	no	"true" (cifrar toda la comunicación de cliente/servidor) "false" (cifrar únicamente la contraseña)	"false"
"server trace"	Especifica el nivel de rastreo del trabajo servidor JDBC. Cuando el rastreo está habilitado, el rastreo empieza al conectarse el cliente al servidor y finaliza al desconectarse la conexión. Debe iniciar el rastreo antes de conectarse al servidor, ya que el cliente habilita el rastreo del servidor únicamente en el momento de la conexión.	no	"0" (el rastreo no está activo) "2" (iniciar el supervisor de base de datos en el trabajo servidor JDBC) "4" (iniciar la depuración en el trabajo servidor JDBC) "8" (guardar las anotaciones de trabajo al finalizar el trabajo servidor JDBC) "16" (iniciar el rastreo de trabajo en el trabajo servidor JDBC) "32" (guardar la información de SQL) Pueden iniciarse varios rastreos agrupando estos valores. Por ejemplo, "6" inicia el supervisor de base de datos e inicia la depuración.	"0"
"thread used"	Especifica si hay que usar hebras en una comunicación con los servidores de sistema principal.	no	"true" "false"	"true"

"trace"	Especifica si deben anotarse mensajes de rastreo. Los mensajes de rastreo son útiles para depurar programas que efectúan llamadas a JDBC. Sin embargo, la anotación de mensajes de rastreo trae aparejado un deterioro del rendimiento, por lo que se recomienda establecer esta propiedad en "true" únicamente para llevar a cabo la depuración. Los mensajes de rastreo se anotan en System.out.	no	"true" "false"	"false"
"translate binary"	Especifica si se convierten los datos binarios. Si esta propiedad se establece en "true", los campos BINARY y VARBINARY se tratan como campos CHAR y VARCHAR.	no	"true" "false"	"false"

Clase AS400JDBCBlob

Puede utilizar un objeto AS400JDBCBlob para acceder a objetos binarios de gran tamaño (BLOB), como por ejemplo los archivos de sonido (.wav) o los archivos de imagen (.gif).

La diferencia clave entre la clase AS400JDBCBlob y la clase AS400JDBCBlobLocator es el lugar de almacenamiento del blob. Con la clase AS400JDBCBlob, el blob se almacena en la base de datos, que infla el tamaño del archivo de base de datos. La clase AS400JDBCBlobLocator almacena en el archivo de base de datos un localizador (es como si fuese un puntero) que señala al lugar en el que se encuentra el blob.

Con la clase AS400JDBCBlob, puede utilizarse la propiedad de umbral de lob. Esta propiedad especifica el tamaño máximo (en kilobytes) de LOB (objeto de gran tamaño) que puede recuperarse como parte de un conjunto de resultados. Los LOB cuyo tamaño sobrepasa este umbral se recuperan en fragmentos, utilizando una comunicación adicional con el servidor. Los umbrales de LOB de mayor tamaño reducen la frecuencia de la comunicación con el servidor, pero bajan más datos de LOB, aunque dichos datos no se utilicen. Los umbrales de LOB de menor tamaño pueden incrementar la frecuencia de la comunicación con el servidor, pero únicamente bajan los datos de LOB según se necesiten. En Propiedades de JDBC encontrará información sobre las propiedades adicionales que están disponibles.

Mediante la clase AS400JDBCBlob se pueden realizar estas tareas:

- Devolver el blob entero como una corriente de bytes sin interpretar
- Devolver parte del contenido del blob
- Devolver la longitud del blob
- »Crear una corriente de datos binarios para escribir en el blob«
- »Escribir una matriz de bytes en el blob«
- »Escribir la totalidad o una parte de una matriz de bytes en el blob«
- »Truncar el blob«

»Ejemplos

Ejemplo: cómo se utiliza la clase AS400JDBCBlob para leer información de un blob:

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

Ejemplo: cómo se utiliza la clase AS400JDBCBlob para actualizar un blob:

```
     ResultSet rs = statement.executeQuery ("SELECT BLOB FROM MYTABLE");
     rs.absolute(5);
     Blob blob = rs.getBlob(1);
          // Cambie los bytes del blob, empezando por el byte 7
          // del blob.
     blob.setBytes (7, new byte[] { (byte) 57, (byte) 58, (byte) 98});
          // Actualice el blob en el conjunto de resultados, cambiando el
blob
          // que empieza en el byte 7 del blob (basado en 1) y truncando el
          // blob al final de los bytes actualizados (ahora el blob tiene 9
bytes).
     rs.updateBlob(1, blob);
          // Actualice la base de datos con el cambio. Así se cambiará el
blob
```

```
            // en la base de datos empezando en el byte 7 del blob y
            // truncando el blob al final de los bytes actualizados.
           rs.updateRow ();
       rs.close();
```

«

Clase AS400JDBCBlobLocator

Puede utilizar un objeto AS400JDBCBlobLocator para acceder a objetos binarios de gran tamaño.

Mediante la clase AS400JDBCBlobLocator se pueden realizar estas tareas:

- Devolver el blob entero como una corriente de bytes sin interpretar
- Devolver parte del contenido del blob
- Devolver la longitud del blob
- »Crear una corriente de datos binarios para escribir en el blob«
- »Escribir una matriz de bytes en el blob«
- »Escribir la totalidad o una parte de una matriz de bytes en el blob«
- »Truncar el blob«

Interfaz CallableStatement

Puede utilizar un objeto CallableStatement para ejecutar procedimientos almacenados de SQL. El procedimiento almacenado al que se llama debe estar ya almacenado en la base de datos. El objeto CallableStatement no contiene el procedimiento almacenado, sino que únicamente llama a dicho procedimiento.

Un procedimiento almacenado puede devolver uno o varios objetos ResultSet y utilizar parámetros IN, OUT e INOUT. Utilice Connection.prepareCall() para crear objetos CallableStatement nuevos.

El objeto CallableStatement permite someter a una base de datos varios mandatos SQL como si fuesen un solo grupo mediante el uso del soporte de proceso por lotes. Puede obtener un mejor rendimiento empleando el soporte de proceso por lotes ya que normalmente se tarda menos en procesar un grupo de operaciones que en procesarlas una a una. Para obtener más información sobre el uso del soporte de proceso por lotes, consulte las mejoras efectuadas en el soporte JDBC.

» CallableStatement permite obtener y establecer parámetros y columnas por nombre, aunque el uso del índice de columna permite obtener un mejor rendimiento. «

El ejemplo que sigue muestra cómo se utiliza la interfaz CallableStatement.

```
    // Conéctese al servidor.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

    // Cree el objeto CallableStatement.
    // Este objeto hace que la llamada
    // especificada se precompile en un
    // procedimiento almacenado. Los signos
    // de interrogación indican dónde se
    // han de establecer los parámetros de
    // entrada, y dónde pueden recuperarse
    // los parámetros de salida. Los dos
    // primeros parámetros son de entrada,
    // y el tercero es un parámetro de salida.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

    // Establezca parámetros de entrada.
cs.setInt (1, 123);
cs.setInt (2, 234);

    // Registre el tipo del
    // parámetro de salida.
cs.registerOutParameter (3, Types.INTEGER);

    // Ejecute el procedimiento almacenado.
cs.execute ();

    // Obtenga el valor del
    // parámetro de salida.
int sum = cs.getInt (3);

    // Cierre el objeto CallableStatement y
    // el objeto Connection.
cs.close();
c.close();
```

Clase AS400JDBCClob

Puede utilizar un objeto AS400JDBCClob para acceder a los objetos de tipo carácter de gran tamaño (CLOB), como por ejemplo los grandes documentos.

La diferencia clave entre la clase AS400JDBCClob y la clase AS400JDBCClobLocator es el lugar de almacenamiento del clob. Con la clase AS400JDBCClob, el clob se almacena en la base de datos, que infla el tamaño del archivo de base de datos. La clase AS400JDBCClobLocator almacena en el archivo de base de datos un localizador (es como si fuese un puntero) que señala al lugar en el que se encuentra el clob.

Con la clase AS400JDBCClob, puede utilizar la propiedad de umbral de lob. Esta propiedad especifica el tamaño máximo (en kilobytes) de LOB (objeto de gran tamaño) que puede recuperarse como parte de un conjunto de resultados. Los LOB cuyo tamaño sobrepasa este umbral se recuperan en fragmentos, utilizando una comunicación adicional con el servidor. Los umbrales de LOB de mayor tamaño reducen la frecuencia de la comunicación con el servidor, pero bajan más datos de LOB, aunque dichos datos no se utilicen. Los umbrales de LOB de menor tamaño pueden incrementar la frecuencia de la comunicación con el servidor, pero únicamente bajan los datos de LOB según se necesiten. En Propiedades de JDBC encontrará información sobre las propiedades adicionales que están disponibles.

Mediante la clase AS400JDBCClob se pueden realizar estas tareas:

- Devolver el clob entero como una corriente de caracteres ASCII
- Devolver el contenido del clob como una corriente de caracteres
- Devolver una parte del contenido del clob
- Devolver la longitud del clob
- »Crear una corriente de caracteres Unicode o una corriente de caracteres ASCII para escribir en el clob«
- »Escribir una serie en el clob«
- »Truncar el clob«

»Ejemplos

Ejemplo: cómo se utiliza la clase AS400JDBCClob para leer información de un clob:

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

Ejemplo: cómo se utiliza la clase AS400JDBCClob para actualizar un clob:

```
     ResultSet rs = statement.executeQuery ("SELECT CLOB FROM MYTABLE");
     rs.absolute(4);
Clob clob = rs.getClob (1);
          // Cambie los caracteres del clob, empezando por el carácter 3
          // del clob.
     clob.setString (3, "Small");
          // Actualice el clob en el conjunto de resultados, empezando por
el
          // carácter 3 del clob y truncando el clob al final de la serie de
          // actualización (el clob ahora tiene 7 caracteres).
     rs.updateClob(1, clob);
          // Actualice la base de datos con el clob actualizado. Así se
cambiará el
          // clob en la base de datos empezando en el carácter 3 del clob y
```

```
            // truncando el clob al final de la serie de actualización.
           rs.updateRow ();
        rs.close();
```

«

Clase AS400JDBCClobLocator

Puede utilizar un objeto AS400JDBCClobLocator para acceder a los objetos de tipo carácter de gran tamaño (CLOB).

Mediante la clase AS400JDBCClobLocator se pueden realizar estas tareas:

- Devolver el clob entero como una corriente de caracteres ASCII
- Devolver el clob entero como una corriente de caracteres
- Devolver una parte del contenido del clob
- Devolver la longitud del clob
- »Crear una corriente de caracteres Unicode o una corriente de caracteres ASCII para escribir en el clob«
- »Escribir una serie en el clob«
- »Truncar el clob«

Clase AS400JDBCConnection

La clase AS400JDBCConnection proporciona una conexión JDBC con una base de datos DB2 UDB para iSeries específica. Utilice DriverManager.getConnection() para crear nuevos objetos AS400JDBCConnection. Para obtener más información, consulte AS400JDBCDriver.

Hay muchos parámetros opcionales que pueden especificarse al crearse la conexión. Las propiedades pueden especificarse como parte del URL o en un objeto java.util.Properties. En Propiedades de JDBC encontrará una lista completa de las propiedades soportadas por AS400JDBCDriver.

Nota: una conexión puede contener como máximo 9999 sentencias abiertas.

AS400JDBCConnection incluye soporte para los puntos de salvar y la posibilidad de retención a nivel de las sentencias, así como soporte limitado para la devolución de claves generadas automáticamente. Para obtener más información sobre estas y otras mejoras, consulte las mejoras efectuadas en el soporte JDBC de Toolbox para Java.

Para utilizar los tickets de kerberos, establezca únicamente el nombre de sistema (y no la contraseña) en el objeto URL JDBC. La identidad del usuario se recupera mediante la infraestructura JGSS (Java Generic Security Services), por lo que tampoco necesita especificar un usuario en el URL JDBC. Sólo puede establecer un método de autenticación en un objeto AS400JDBCConnection a la vez. Al establecer la contraseña se borran los tickets de kerberos o los símbolos de perfil. Para obtener más información, consulte la clase AS400 y la documentación de seguridad de J2SDK v1.4.

Mediante la clase AS400JDBCConnection se pueden realizar estas tareas:

- Crear una sentencia (objetos Statement, PreparedStatement o CallableStatement)
- Crear una sentencia que tenga un tipo de conjunto de resultados y una concurrencia específicos (objetos Statement, PreparedStatement o CallableStatement)
- Comprometer y retrotraer los cambios efectuados en la base de datos y liberar los bloqueos de base de datos que están retenidos actualmente
- Cerrar la conexión y cerrar los recursos del servidor de inmediato en lugar de esperar a que se liberen automáticamente
- Establecer la posibilidad de retención y obtener la posibilidad de retención de la conexión
- Establecer el aislamiento de transacción y obtener el aislamiento de transacción de la conexión
- Obtener los metadatos de la conexión
- Activar o desactivar el compromiso automático
- Obtener el identificador de trabajo del trabajo servidor de sistema principal que corresponde a la conexión

Si utiliza JDBC 3.0 y se conecta a un servidor ejecutando OS/400 V5R2, puede emplear AS400JDBCConnection para llevar a cabo las acciones siguientes:

- Crear una sentencia con una posibilidad de retención de conjunto de resultados específica (objeto Statement, PreparedStatement o CallableStatement)
- Crear una sentencia preparada que devuelva las claves generadas automáticamente (cuando se llama a getGeneratedKeys() en el objeto Statement)
- Utilizar puntos de salvar, que ofrecen un control más específico de las transacciones:
    - Establecer puntos de salvar
    - Retrotraer puntos de salvar
    - Liberar puntos de salvar

AS400JDBCConnectionPool

La clase AS400JDBCConnectionPool representa una agrupación de objetos AS400JDBCConnection disponibles para su uso por un programa Java como parte del soporte de la Caja de Herramientas para la API JDBC 2.0 Optional Package.

Puede emplear un objeto AS400JDBCConnectionPoolDataSource para especificar las propiedades de las conexiones que se crean en la agrupación, como se muestra en el ejemplo siguiente.

No podrá cambiar el origen de datos de la agrupación de conexiones una vez que haya solicitado una conexión y la agrupación esté en uso. Para restablecer el origen de datos de la agrupación de conexiones, primero debe efectuar una llamada a close() en la agrupación.

Para devolver conexiones a un objeto AS400JDBCConnectionPool, utilice close() en el objeto AS400JDBCConnection.

Nota: si las conexiones no se devuelven a la agrupación, el tamaño de la agrupación de conexiones sigue creciendo y las conexiones no se reutilizan.

Para establecer propiedades en la agrupación, utilice los métodos heredados de ConnectionPool. Entre las propiedades que puede establecer se encuentran las siguientes:

- Número máximo de conexiones permitidas en la agrupación
- Tiempo máximo de vida de una conexión
- Tiempo máximo de inactividad de una conexión

También puede registrar objetos AS400JDBCConnectionPoolDataSource mediante un proveedor de servicio JNDI (Java Naming and Directory Interface(TM)). Para obtener más información acerca de los proveedores de servicio JNDI, consulte IBM Toolbox para Java - Enlaces de consulta o referencia.

Ejemplo: cómo se utiliza la agrupación de conexiones

El ejemplo siguiente obtiene de JNDI un origen de datos de agrupación de conexiones y lo emplea para crear una agrupación de conexiones con 10 conexiones:

```
          // Obtenga un objeto AS400JDBCConnectionPoolDataSource de JNDI
          // (se supone que el entorno JNDI está establecido).
     Context context = new InitialContext(environment);
     AS400JDBCConnectionPoolDataSource datasource =

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

     // Cree un objeto AS400JDBCConnectionPool.
     AS400JDBCConnectionPool pool = new AS400JDBCConnectionPool(datasource);

          // Añada 10 conexiones a la agrupación que la aplicación pueda
          // utilizar (las conexiones físicas de base de datos se crean
          // según el origen de datos).
     pool.fill(10);

         // Obtenga un handle con una conexión de base de datos de la
agrupación.
     Connection connection = pool.getConnection();

     ...Realizar diversas consultas/actualizaciones en la base de datos.
```

```
    // Cierre el handle de la conexión para devolverla a la agrupación.
    connection.close();

...La aplicación trabaja con varias aplicaciones más de la agrupación.

    // Cierre la agrupación para liberar todos los recursos.
    pool.close();
```

Interfaz DatabaseMetaData

Puede utilizar un objeto DatabaseMetaData para obtener información acerca de la base de datos como conjunto, y obtener asimismo información de catálogo.

El siguiente ejemplo muestra cómo se obtiene una lista de tablas, que es una función de catálogo:

```
    // Conéctese al servidor.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

    // Obtenga los metadatos de base de datos
    // de la conexión.
DatabaseMetaData dbMeta = c.getMetaData();

    // Obtenga una lista de tablas que coincidan
    // con los criterios siguientes.
String catalog = "myCatalog";
String schema  = "mySchema";
String table   = "myTable%"; // % indica el patrón de búsqueda
String types[]  = {"TABLE", "VIEW", "SYSTEM TABLE"};
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

    // ...Itere por ResultSet
    // para obtener los valores.

    // Cierre el objeto Connection.
c.close();
```

Clase AS400JDBCDataSource

La clase AS400JDBCDataSource representa una fábrica de conexiones de base de datos de iSeries. La clase AS400JDBCConnectionPoolDataSource representa una fábrica de objetos AS400JDBCPooledConnection.

Puede registrar cualquier tipo de objeto de origen de datos mediante un proveedor de servicio JNDI (Java Naming and Directory Interface). Para obtener más información acerca de los proveedores de servicio JNDI, consulte IBM Toolbox para Java - Enlaces de consulta o referencia.

Ejemplos

Los ejemplos siguientes muestran formas de crear y utilizar objetos AS400JDBCDataSource. Los dos últimos ejemplos muestran cómo registrar un objeto AS400JDBCDataSource en JNDI y a continuación emplear el objeto devuelto de JNDI para obtener una conexión de base de datos. Observe que incluso al usar distintos proveedores de servicio JNDI, el código es muy parecido.

Ejemplo: crear un objeto AS400JDBCDataSource

A continuación figura un ejemplo en el que se crea un objeto AS400JDBCDataSource y se conecta éste con una base de datos:

```
        // Cree un origen de datos para efectuar la conexión.
        AS400JDBCDataSource datasource = new AS400JDBCDataSource("myAS400");
        datasource.setUser("myUser");
        datasource.setPassword("MYPWD");

        // Cree una conexión de base de datos con el iSeries.
        Connection connection = datasource.getConnection();
```

Ejemplo: crear un objeto AS400JDBCConnectionPoolDataSource que pueda utilizarse para almacenar en la antememoria conexiones JDBC

El ejemplo siguiente muestra cómo utilizar un objeto AS400JDBCConnectionPoolDataSource para almacenar en la antememoria conexiones JDBC.

```
        // Cree un origen de datos para efectuar la conexión.
        AS400JDBCConnectionPoolDataSource dataSource = new
AS400JDBCConnectionPoolDataSource("myAS400");
        datasource.setUser("myUser");
        datasource.setPassword("MYPWD");

        // Obtenga PooledConnection.
        PooledConnection pooledConnection = datasource.getPooledConnection();
```

Ejemplo: cómo se utilizan las clases de proveedor de servicio JNDI para almacenar un objeto AS400JDBCDataSource

El ejemplo siguiente muestra cómo utilizar las clases de proveedor de servicio JNDI para almacenar un objeto DataSource directamente en el sistema de archivos IFS del servidor:

```
        // Cree un origen de datos para la base de datos del iSeries.
        AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
        dataSource.setServerName("myAS400");
        dataSource.setDatabaseName("myAS400 Database");

        // Registre el origen de datos en JNDI (Java Naming and Directory
Interface).
        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);

       // Devuelva un objeto AS400JDBCDataSource de JNDI y obtenga una
conexión.
       AS400JDBCDataSource datasource = (AS400JDBCDataSource)
context.lookup("jdbc/customer");
       Connection connection = datasource.getConnection("myUser", "MYPWD");
```

Ejemplo: cómo se utilizan los objetos AS400JDBCDataSource y las clases de IBM SecureWay Directory con un servidor de directorios LDAP (Lightweight Directory Access Protocol)

El ejemplo siguiente muestra cómo utilizar las clases de IBM SecureWay Directory para almacenar un objeto en un servidor de directorios LDAP (Lightweight Directory Access Protocol):

```
       // Cree un origen de datos para la base de datos del iSeries.
       AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
       dataSource.setServerName("myAS400");
       dataSource.setDatabaseName("myAS400 Database");

       // Registre el origen de datos en JNDI (Java Naming and Directory
Interface).
       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);

       // Devuelva un objeto AS400JDBCDataSource de JNDI y obtenga una
conexión.
       AS400JDBCDataSource datasource = (AS400JDBCDataSource)
context.lookup("jdbc/customer");
            cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion");
       Connection connection = datasource.getConnection("myUser", "MYPWD");
```

Registro del controlador JDBC

Antes de utilizar JDBC para acceder a los datos de un archivo de base de datos del servidor, es necesario registrar el controlador JDBC para el programa bajo licencia IBM Toolbox para Java en DriverManager. El controlador se puede registrar ya sea utilizando una propiedad de sistema Java o haciendo que el programa Java registre el controlador:

- Registrar mediante una propiedad del sistema

  Cada máquina virtual tiene su propio método para establecer las propiedades del sistema. Por ejemplo, el mandato Java de JDK utiliza la opción -D para establecer las propiedades del sistema. Para establecer el controlador mediante las propiedades del sistema, especifique:

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

- » Registrar mediante el programa Java

  Para cargar el controlador JDBC de Toolbox para Java, antes de la primera llamada a JDBC, añada al programa Java la línea siguiente:

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

  El controlador JDBC de Toolbox para Java se registra cuando se carga (éste es el método preferido de registrar el controlador). También puede registrar explícitamente el controlador JDBC de la Caja de Herramientas con lo siguiente:

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

  «

El controlador JDBC de IBM Toolbox para Java no requiere un objeto AS400 como parámetro de entrada, como lo requieren las demás clases de IBM Toolbox para Java que obtienen datos de un servidor. Sin embargo, internamente sí que se utiliza un objeto AS400 para gestionar el usuario por omisión y la colocación de contraseñas en antememoria. Cuando se establece una conexión con el servidor por primera vez, puede solicitarse al usuario que escriba el ID de usuario y la contraseña. El usuario puede optar por guardar el ID de usuario como ID de usuario por omisión y añadir la contraseña a la antememoria de contraseñas. Al igual que en las demás funciones de IBM Toolbox para Java, si es el programa Java el que proporciona el ID de usuario y la contraseña, el usuario por omisión no se establece y la contraseña no se pone en la antememoria. En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones.

Utilización del controlador JDBC para conectarse a una base de datos del servidor

El método DriverManager.getConnection() le permite conectarse a la base de datos del servidor. DriverManager.getConnection() toma como argumento una serie de URL (localizador uniforme de recursos). El gestor de controladores JDBC intenta localizar un controlador que pueda conectarse a la base de datos representada por el URL. Cuando utilice el controlador de IBM Toolbox para Java, emplee la siguiente sintaxis para el URL:

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

Nota: en el URL se puede omitir systemName o defaultSchema.

» Para utilizar los tickets de kerberos, establezca únicamente el nombre de sistema (y no la contraseña) en el objeto URL JDBC. La identidad del usuario se recupera mediante la infraestructura JGSS (Java Generic Security Services), por lo que tampoco necesita especificar un usuario en el URL JDBC. Sólo puede establecer un método de autenticación en un objeto AS400JDBCConnection a la vez. Al establecer la contraseña se borran los tickets de kerberos o los símbolos de perfil. Para obtener más información, consulte la clase AS400 y la documentación de

seguridad de J2SDK v1.4  .«

Ejemplos: utilización del controlador JDBC para conectarse a un servidor

Ejemplo 1: utilizar un URL en el que no se especifica un nombre de sistema. Ello hace que se solicite al usuario que escriba el nombre del sistema al que desea conectarse.

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

Ejemplo 2: conectarse a la base de datos del servidor; no se especifica ningún esquema por omisión ni ninguna propiedad.

```
        // Conéctese al sistema 'mySystem'.
        // No se especifica ningún esquema por
        // omisión ni ninguna propiedad.
    Connection c = DriverManager.getConnection("jdbc:as400://mySystem");
```

Ejemplo 3: conectarse a la base de datos del servidor; se especifica un esquema por omisión.

```
        // Conéctese al sistema 'mySys2'.
        // Se especifica el esquema por
        // omisión 'myschema'.
   Connection c2 =
DriverManager.getConnection("jdbc:as400://mySys2/mySchema");
```

Ejemplo 4: conectarse a la base de datos del servidor; se especifican propiedades mediante java.util.Properties. El programa Java puede especificar un conjunto de propiedades de JDBC ya sea utilizando la interfaz java.util.Properties o especificando dichas propiedades como parte del URL. En Propiedades de JDBC encontrará una lista de las propiedades soportadas.

Por ejemplo, para especificar las propiedades mediante la interfaz Properties, podría utilizar el siguiente código:

```
        // Cree un objeto propiedades.
    Properties p = new Properties();

        // Establezca las propiedades para
        // la conexión.
    p.put("naming", "sql");
    p.put("errors", "full");

        // Conéctese utilizando el
        // objeto de propiedades.
    Connection c = DriverManager.getConnection("jdbc:as400://mySystem",p);
```

Ejemplo 5: conectarse a la base de datos del servidor; las propiedades se especifican mediante un URL (localizador uniforme de recursos).

```
        // Conéctese utilizando las propiedades.
        // Éstas se establecen en el URL,
        // en vez de establecerse mediante
        // un objeto de propiedades.
    Connection c = DriverManager.getConnection(
                       "jdbc:as400://mySystem;naming=sql;errors=full");
```

Ejemplo 6: conectarse a la base de datos del servidor; se especifican el ID de usuario y la contraseña.

```
        // Conéctese utilizando las propiedades
```

```
    // indicadas en el URL y especificando
    // un ID de usuario y una contraseña.
Connection c = DriverManager.getConnection(
                  "jdbc:as400://mySystem;naming=sql;errors=full",
                  "auser",
                  "apassword");
```

Ejemplo 7: desconectarse de la base de datos. Utilice el método close() del objeto Connecting para desconectarse del servidor. Para cerrar la conexión creada en el ejemplo anterior, utilice esta sentencia:

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

AS400JDBCParameterMetaData

La clase AS400JDBCParameterMetaData permite a los programas recuperar información sobre las propiedades de parámetros de los objetos PreparedStatement y CallableStatement.

AS400JDBCParameterMetaData ofrece métodos que permiten llevar a cabo las acciones siguientes:

- Obtener el nombre de clase del parámetro
- Obtener el número de parámetros de PreparedStatement
- Obtener el tipo SQL del parámetro
- Obtener el nombre de tipo específico de la base de datos del parámetro
- Obtener la precisión o la escala del parámetro

Ejemplo: cómo se utiliza AS400JDBCParameterMetaData

En el ejemplo siguiente se muestra cómo se utiliza AS400JDBCParameterMetaData para recuperar parámetros de un objeto PreparedStatement generado dinámicamente:

```
// Obtenga una conexión desde el controlador.
Class.forName("com.ibm.as400.access.AS400JDBCDriver");
Connection connection =
   DriverManager.getConnection
      ("jdbc:as400://myAS400",
       "myUserId",
       "myPassword");
// Cree un objeto de sentencia preparada.
PreparedStatement ps =
   connection.prepareStatement
      ("SELECT STUDENTS FROM STUDENTTABLE WHERE STUDENT_ID= ?");
// Establezca un ID de alumno en el parámetro 1.
ps.setInt(1, 123456);
// Recupere los metadatos de parámetro de la sentencia preparada.
ParameterMetaData pMetaData = ps.getParameterMetaData();
// Recupere el número de parámetros de la sentencia preparada.
// Devuelve 1.
int parameterCount = pMetaData.getParameterCount();
// Averigüe el nombre de tipo de parámetro del parámetro 1.
// Devuelve INTEGER.
String getParameterTypeName = pMetaData.getParameterTypeName(1);
```

Interfaz PreparedStatement

Puede utilizar un objeto PreparedStatement cuando vea que una sentencia SQL se va a ejecutar muchas veces. Una sentencia SQL puede precompilarse. Una sentencia "preparada" es una sentencia SQL que se ha precompilado. Esta solución es más eficaz que la de ejecutar muchas veces la misma sentencia utilizando un objeto Statement, pues de esta última forma la sentencia se compila cada vez que se ejecuta. Además, la sentencia SQL contenida en un objeto PreparedStatement puede tener uno o varios parámetros IN. Utilice Connection.prepareStatement() para crear objetos PreparedStatement.

El objeto PreparedStatement permite someter a una base de datos varios mandatos SQL como si fuesen un solo grupo mediante el uso del soporte de proceso por lotes. Puede obtener un mejor rendimiento empleando el soporte de proceso por lotes ya que normalmente se tarda menos en procesar un grupo de operaciones que en procesarlas una a una. Para obtener más información sobre el uso del soporte de proceso por lotes, consulte las mejoras efectuadas en el soporte JDBC.

El ejemplo que sigue muestra cómo se utiliza la interfaz PreparedStatement.

```
        // Conéctese al servidor.
    Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

        // Cree el objeto PreparedStatement.
        // Este objeto precompila la
        // sentencia SQL especificada. Los
        // signos de interrogación indican
        // dónde se han de establecer parámetros
        // antes de que se ejecute la sentencia.
    PreparedStatement ps = c.prepareStatement("INSERT INTO
MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

        // Establezca parámetros y ejecute
        // la sentencia.
    ps.setString(1, "JOSH");
    ps.setInt(2, 789);
    ps.executeUpdate();

        // Establezca parámetros y ejecute
        // la sentencia de nuevo.
    ps.setString(1, "DAVE");
    ps.setInt(2, 456);
    ps.executeUpdate();

        // Cierre el objeto PreparedStatement y
        // el objeto Connection.
    ps.close();
    c.close();
```

ResultSet

Puede utilizar un objeto ResultSet para acceder a una tabla de datos generada al ejecutar una consulta. Las filas de la tabla se recuperan en secuencia. Dentro de una fila, es posible acceder a los valores de las columnas en cualquier orden.

Los datos almacenados en ResultSet se recuperan mediante los diversos métodos get, en función del tipo de datos que se vaya a recuperar. El método next() permite desplazarse a la fila siguiente.

»ResultSet permite obtener y actualizar columnas por nombre, aunque el uso del índice de columna permite obtener un mejor rendimiento.«

Movimiento de cursor

Un cursor, que es un puntero interno utilizado por un conjunto de resultados, señala a la fila perteneciente a dicho conjunto y a la que está accediendo el programa Java.

»Se ha mejorado el rendimiento del método getRow(). Antes de la versión V5R2, al utilizar ResultSet.last(), ResultSet.afterLast() y ResultSet.absolute() con un valor negativo, el número de fila actual no estaba disponible. Las restricciones anteriores se han eliminado, con lo que el método getRow() es ahora totalmente funcional.«

JDBC 2.0 y las especificaciones de JDBC posteriores proporcionan métodos adicionales para acceder a posiciones específicas dentro de una base de datos:

Posiciones de cursor desplazables	
absolute afterLast beforeFirst first getRow isAfterLast isBeforeFirst	isFirst isLast last moveToCurrentRow moveToInsertRow previous relative

Posibilidades de desplazamiento

Si un conjunto de resultados se crea mediante la ejecución de una sentencia, es posible moverse (desplazarse) por las filas de una tabla en sentido hacia atrás (de la última a la primera) o hacia delante (de la primera a la última).

Un conjunto de resultados que dé soporte a este movimiento se llama desplazable. Los conjuntos de resultados desplazables admiten también el posicionamiento relativo y el absoluto. El posicionamiento relativo le permite moverse a una fila del conjunto de resultados especificando una posición relativa a la fila actual. El posicionamiento absoluto le permite moverse directamente a una fila especificando la posición que dicha fila tiene en el conjunto de resultados.

Con JDBC 2.0 y las especificaciones de JDBC posteriores, se dispone de dos posibilidades de desplazamiento adicionales al trabajar con la clase ResultSet: conjuntos de resultados no sensibles al desplazamiento y sensibles al desplazamiento.

Un conjunto de resultados no sensible al desplazamiento suele no ser sensible a los cambios realizados mientras está abierto, mientras que el conjunto de resultados sensible al desplazamiento es sensible a los cambios.»El controlador JDBC de IBM Toolbox para Java no soporta conjuntos de resultados no sensibles al desplazamiento«

Conjuntos de resultados actualizables

En la aplicación, puede utilizar conjuntos de resultados que emplean ya sea una concurrencia de solo lectura (no pueden realizarse actualizaciones en los datos) o una concurrencia actualizable (permite realizar actualizaciones en los datos y puede utilizar bloqueos de escritura de base de datos para controlar las distintas transacciones que acceden a un mismo elemento de datos). En un conjunto de resultados actualizable, las filas se pueden actualizar, insertar y suprimir. Puede disponer de numerosos métodos de actualización para usarlos en el programa; por ejemplo, puede:

- Actualizar una corriente de datos ASCII
- Actualizar BigDecimal
- Actualizar una corriente de datos binarios

En Resumen de métodos encontrará una lista completa de los métodos de actualización que están disponibles mediante la interfaz ResultSet.

Ejemplo: conjuntos de resultados actualizables

El ejemplo que figura a continuación muestra cómo se utiliza un conjunto de resultados que permite realizar actualizaciones en los datos (concurrencia de actualización) y realizar cambios en el conjunto de resultados mientras permanece abierto (sensible al desplazamiento).

```
        // Conéctese al servidor.
     Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

        // Cree un objeto Statement. Establezca la concurrencia
        // del conjunto de resultados en actualizable.
     Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_UPDATABLE);

        // Ejecute una consulta. El resultado se coloca
        // en un objeto ResultSet.
     ResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE
FOR UPDATE");

        // Itere por las filas de ResultSet.
        // A medida que lea la fila, vaya
        // actualizándola con un nuevo ID.
     int newId = 0;
     while (rs.next ())
     {

        // Obtenga los valores de ResultSet.
        // El primer valor es una serie y
        // el segundo valor es un entero.
        String name = rs.getString("NAME");
        int id = rs.getInt("ID");

        System.out.println("Nombre = " + name);
        System.out.println("ID antiguo = " + id);

        // Actualice el ID con un nuevo entero.
        rs.updateInt("ID", ++newId);

        // Envíe las actualizaciones al servidor.
        rs.updateRow ();

        System.out.println("ID nuevo = " + newId);
```

```
    }

        // Cierre el objeto Statement y
        // el objeto Connection.
    s.close();
    c.close();
```

ResultSetMetaData

La interfaz ResultSetMetaData determina los tipos y las propiedades de las columnas de un conjunto de resultados (ResultSet).

»Al conectarse a un servidor que ejecuta OS/400 V5R2 o posterior, el uso de la propiedad de metadatos ampliados permite aumentar la precisión de los siguientes métodos de ResultSetMetaData:

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

Asimismo, al establecer esta propiedad en true se habilita el soporte para el método ResultSetMetaData.getSchemaName(int). Tenga en cuenta que al utilizar la propiedad de metadatos ampliados puede reducirse el rendimiento ya que es preciso recuperar más información del servidor.«

AS400JDBCRowSet

La clase AS400JDBCRowSet representa un conjunto de filas conectado que encapsula un conjunto de resultados JDBC. Los métodos de AS400JDBCRowSet son muy parecidos a los de AS400JDBCResultSet. La conexión con la base de datos se mantiene mientras está en uso.

Puede utilizar un objeto AS400JDBCDataSource o un objeto AS400JDBCConnectionPoolDataSource para crear la conexión con la base de datos que desee utilizar para acceder a los datos de AS400JDBCRowSet.

Ejemplos

Los ejemplos que hay a continuación muestran cómo se utiliza la clase AS400JDBCRowSet.

Ejemplo: crear, llenar con datos y actualizar un objeto AS400JDBCRowSet

```
            DriverManager.registerDriver(new AS400JDBCDriver());
       // Establezca conexión mediante un URL.
       AS400JDBCRowSet rowset = new
AS400JDBCRowSet("jdbc:as400://mySystem","myUser", "myPassword");

       // Establezca el mandato utilizado para llenar con datos la lista.
       rowset.setCommand("SELECT * FROM MYLIB.DATABASE");

         // Llene con datos el conjunto de filas.
         rowset.execute();

       // Actualice los saldos de cliente.
       while (rowset.next())
       {
          double newBalance = rowset.getDouble("BALANCE") +
july_statements.getPurchases(rowset.getString("CUSTNUM"));
          rowset.updateDouble("BALANCE", newBalance);
            rowset.updateRow();
       }
```

Ejemplo: crear y llenar con datos un objeto AS400JDBCRowSet, al tiempo que se obtiene el origen de datos de JNDI

```
       // Obtenga el origen de datos que está registrado en JNDI (supongamos
que el entorno JNDI esté establecido).
       Context context = new InitialContext();
       AS400JDBCDataSource dataSource = (AS400JDBCDataSource)
context.lookup("jdbc/customer");

       AS400JDBCRowSet rowset = new AS400JDBCRowSet();
       // Establezca conexión definiendo el nombre de origen de datos.
       rowset.setDataSourceName("jdbc/customer");
       rowset.setUsername("myuser");
       rowset.setPassword("myPasswd");

       // Establezca la sentencia preparada e inicialice los parámetros.
       rowset.setCommand("SELECT * FROM MYLIBRARY.MYTABLE WHERE STATE = ?
AND BALANCE > ?");
       rowset.setString(1, "MINNESOTA");
       rowset.setDouble(2, MAXIMUM_LIMIT);

         // Llene con datos el conjunto de filas.
```

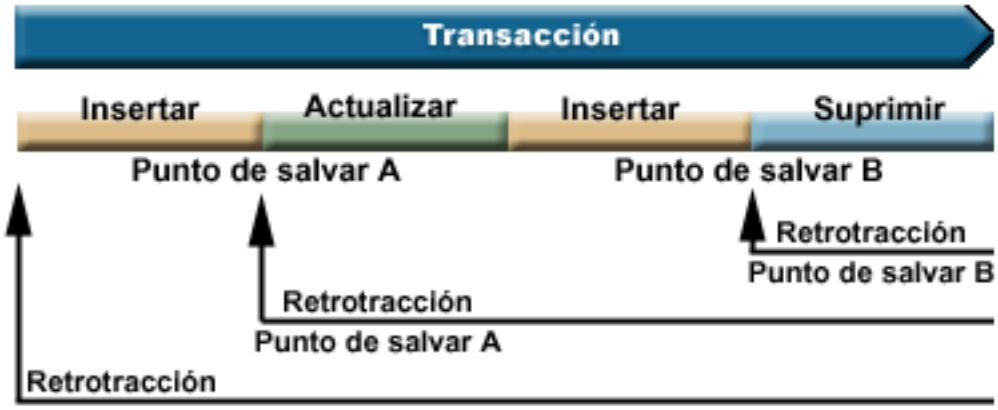
```
rowset.execute();
```

»

Clase AS400JDBCSavepoint

La clase AS400JDBCSavepoint representa un punto de interrupción lógico en una transacción. El uso de los puntos de salvar permite controlar de modo más preciso los cambios a los que afecta la retrotracción de una transacción.

Figura 1: cómo se utilizan los puntos de salvar para controlar las retrotracciones en una transacción



Por ejemplo, la figura 1 muestra una transacción que contiene dos puntos de salvar, A y B. Al retrotraer la transacción a cualquiera de los dos puntos de salvar únicamente se deshacen los cambios desde el punto en que se llama a una retrotracción hasta el punto de salvar. De este modo se evita tener que deshacer todos los cambios de toda la transacción. Observe que, una vez efectuada la retrotracción hasta el punto de salvar A, posteriormente no puede efectuar la retrotracción hasta el punto de salvar B. No puede acceder al punto de salvar B después de haberse retrotraído el trabajo más allá del mismo.

Ejemplo: cómo se utilizan los puntos de salvar

En este ejemplo, supongamos que su aplicación actualiza los registros de alumnos. Tras actualizar un campo determinado de cada registro de alumno, efectúa una operación de compromiso. El código detecta un error concreto asociado a la actualización de este campo y retrotrae el trabajo realizado cuando se produce este error. Usted sabe que este error concreto afecta únicamente al trabajo efectuado en el registro actual.

Por consiguiente, establece un punto de salvar entre cada actualización de los registros de alumnos. Ahora, cuando se produce este error, sólo retrotrae la última actualización de la tabla de alumnos. En lugar de tener que retrotraer una gran cantidad de trabajo, ahora puede retrotraer únicamente una pequeña cantidad de trabajo.

El código del ejemplo siguiente muestra cómo pueden utilizarse los puntos de salvar. En el ejemplo se supone que el ID de alumno de John es 123456 y el ID de alumno de Jane es 987654.

```
     // Obtenga una conexión desde el controlador.
     Class.forName("com.ibm.as400.access.AS400JDBCDriver");
     // Obtenga un objeto Statement.
         Statement statement = connection.createStatement();
     // Actualice el registro de John con su nota 'B' en gimnasia.
     int rows = statement.executeUpdate("UPDATE STUDENTTABLE SET
GRADE_SECOND_PERIOD = 'B'
            WHERE STUDENT_ID= '123456'");
     // Establezca un punto de salvar marcando un punto intermedio en la
transacción.
     Savepoint savepoint1 = connection.setSavepoint("SAVEPOINT_1");
```

```
     // Actualice el registro de Jane con su nota 'C' en bioquímica.
     int rows = statement.executeUpdate("UPDATE STUDENTTABLE SET
GRADE_SECOND_PERIOD = 'C'
            WHERE STUDENT_ID= '987654'");
     // Se ha detectado un error, por lo que debemos retrotraer el registro
de Jane, pero
     // no el de John. Retrotraiga la transacción hasta el punto de salvar
1. El cambio del
     // registro de Jane se elimina, mientras que el del registro de John se
conserva.
     connection.rollback(savepoint1);
     // Comprometa la transacción; sólo se compromete la nota 'B' de John en
la base de datos.
     connection.commit();
```

Consideraciones y restricciones

Para utilizar los puntos de salvar deben tenerse presentes las consideraciones y restricciones siguientes:

Consideraciones

Toolbox para Java sigue las reglas de base de datos en relación con el modo en que las retrotracciones afectan a los cursores y bloqueos retenidos. Por ejemplo, al establecer la opción de conexión de modo que se mantengan abiertos los cursores tras una retrotracción tradicional, los cursores también permanecen abiertos tras una retrotracción hasta un punto de salvar. Dicho de otro modo, cuando se produce una petición de retrotracción en la que intervienen puntos de salvar, Toolbox para Java no mueve ni cierra el cursor si la base de datos subyacente no soporta esta función.

Al utilizar un punto de salvar para retrotraer una transacción sólo se deshacen las acciones efectuadas desde el punto de inicio de la retrotracción hasta el punto de salvar. Las acciones efectuadas antes de ese punto de salvar se conservan. Como se muestra en el ejemplo anterior, tenga en cuenta que puede comprometer una transacción que contenga trabajo efectuado antes de un punto de salvar determinado pero que no contenga trabajo efectuado tras el punto de salvar

Todos los puntos de salvar se liberan y dejan de ser válidos cuando se compromete la transacción o cuando se retrotrae toda la transacción. También puede liberar puntos de salvar llamando a Connection.releaseSavepoint().

Restricciones

Al utilizar puntos de salvar se aplican las restricciones siguientes:

- Los puntos de salvar con nombre deben ser exclusivos.
- No puede reutilizar un nombre de punto de salvar hasta que se libere, comprometa o retrotraiga el punto de salvar.
- El compromiso automático debe establecerse en 'OFF' para que los puntos de salvar sean válidos. Puede establecer el compromiso automático en 'OFF' mediante `Connection.setAutoCommit(false)`. Si se habilita el compromiso automático cuando se utilizan puntos de salvar se lanza una excepción.
- Los puntos de salvar no son válidos en las conexiones XA. Si se utiliza una conexión XA con puntos de salvar se lanza una excepción.
- El servidor debe ejecutar OS/400 Versión 5 Release 2 o posterior. Si se utilizan puntos de salvar al conectarse (o estando ya conectado) a un servidor que ejecuta la versión V5R1 o anterior de OS/400 se lanza una excepción.

Ejecución de sentencias SQL con objetos Statement

Utilice un objeto Statement para ejecutar una sentencia SQL y, opcionalmente, obtener el conjunto de resultados (ResultSet) generado por ella.

PreparedStatement es heredera de Statement, y CallableStatement es heredera de PreparedStatement. Utilice los siguientes objetos Statement para ejecutar las distintas sentencias SQL:

- Statement - permite ejecutar una sentencia SQL simple que no tiene ningún parámetro.
- PreparedStatement - permite ejecutar una sentencia SQL precompilada que puede tener o no tener parámetros IN.
- CallableStatement - permite ejecutar una llamada a un procedimiento almacenado de base de datos.El objeto CallableStatement puede tener o no tener parámetros IN, OUT e INOUT.

El objeto Statement permite someter a una base de datos varios mandatos SQL como si fuesen un solo grupo mediante el uso del soporte de proceso por lotes. Puede obtener un mejor rendimiento empleando el soporte de proceso por lotes ya que normalmente se tarda menos en procesar un grupo de operaciones que en procesarlas una a una. Para obtener más información sobre el uso del soporte de proceso por lotes, consulte las mejoras efectuadas en el soporte JDBC.

Cuando se utilizan las actualizaciones por lotes, suele ser conveniente desactivar el compromiso automático. La desactivación del compromiso automático permite al programa determinar si debe comprometer la transacción en el caso de que se produzca un error y no se hayan ejecutado todos los mandatos. En JDBC 2.0 y las especificaciones de JDBC posteriores, un objeto Statement puede hacer un seguimiento de una lista de mandatos que pueden someterse satisfactoriamente y ejecutarse conjuntamente en un grupo. Cuando el método executeBatch() ejecuta esta lista de mandatos por lotes, la ejecución de los mandatos se realiza en el orden en que se añadieron a la lista.

AS400JDBCStatement proporciona métodos que permiten llevar a cabo muchas acciones, entre ellas las siguientes:

- Ejecutar distintos tipos de sentencias
- Recuperar los valores de distintos parámetros del objeto Statement, tales como:
  - La conexión
  - Las claves generadas automáticamente que se han creado como consecuencia de la ejecución del objeto Statement
  - El tamaño de búsqueda y la dirección de búsqueda
  - El tamaño máximo de campo y el límite máximo de fila
  - El conjunto de resultados actual, el conjunto de resultados siguiente, el tipo de conjunto de resultados, la concurrencia del conjunto de resultados y la posibilidad de retención del cursor del conjunto de resultados
- Añadir una sentencia SQL al proceso por lotes actual
- Ejecutar el proceso por lotes actual de sentencias SQL

Interfaz Statement

Utilice Connection.createStatement() para crear objetos Statement nuevos.

El ejemplo que sigue muestra cómo se utiliza un objeto Statement.

```
    // Conéctese al servidor.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");
```

```
        // Cree un objeto Statement.
    Statement s = c.createStatement();

        // Ejecute una sentencia SQL que cree
        // una tabla en la base de datos.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID
INTEGER)");

        // Ejecute una sentencia SQL que inserte
        // un registro en la tabla.
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES
('DAVE', 123)");

        // Ejecute una sentencia SQL que inserte
        // un registro en la tabla.
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES
('CINDY', 456)");

        // Ejecute una consulta SQL en la tabla.
    ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

        // Cierre el objeto Statement y
        // el objeto Connection.
    s.close();
    c.close();
```

Gestión de transacciones distribuidas XA de JDBC

Las clases de gestión de transacciones distribuidas XA de JDBC permiten utilizar el controlador JDBC de IBM Toolbox para Java dentro de una transacción distribuida. La utilización de las clases de XA para habilitar el controlador JDBC de IBM Toolbox para Java permite a éste participar en las transacciones que abarcan varios orígenes de datos.

Normalmente las clases de gestión de transacciones distribuidas XA se utilizan y controlan directamente con un gestor de transacciones independiente del controlador JDBC. Las interfaces de gestión de transacciones distribuidas se definen como parte de JDBC 2.0 Optional Package y JTA (Java Transaction API). Ambas están disponibles a través de Sun como archivos jar.»Las interfaces de gestión de transacciones distribuidas también están soportadas en la API JDBC 3.0, que está empaquetada con la plataforma Java 2, Standard Edition, versión 1.4.«

Encontrará más información en los sitios Web de Sun que corresponden a JDBC y JTA.

Utilice los objetos siguientes para permitir que el controlador JDBC de IBM Toolbox para Java participe en las transacciones distribuidas XA:

- AS400JDBCXADataSource - Una fábrica de objetos AS400JDBCXAConnection. Es una subclase de AS400JDBCDataSource.
- AS400JDBCXACConnection - Un objeto de conexión de agrupación que proporciona ganchos para la gestión de agrupaciones de conexiones y la gestión de recursos XA.
- AS400JDBCXAResource - Un gestor de recursos destinado al uso en la gestión de transacciones XA.

Ejemplo: cómo se utilizan las clases de XA

El ejemplo siguiente muestra un uso sencillo de las clases de XA. Tenga en cuenta que los detalles se incluirían con tareas que utilizaran otros orígenes de datos. Este tipo de código habitualmente aparece dentro de un gestor de transacciones.

```
     // Cree un origen de datos XA para establecer la conexión XA.
     AS400JDBCXADataSource xaDataSource = new
AS400JDBCXADataSource("myAS400");
     xaDataSource.setUser("myUser");
     xaDataSource.setPassword("myPasswd");

     // Obtenga un objeto XAConnection y obtenga el objeto XAResource
asociado.
     // Esto proporciona acceso al gestor de recursos.
     XAConnection xaConnection = xaDataSource.getXAConnection();
     XAResource xaResource = xaConnection.getXAResource();

     // Genere un nuevo Xid (esta tarea corresponde al gestor de
transacciones).
     Xid xid = ...;

     // Inicie la transacción.
     xaResource.start(xid, XAResource.TMNOFLAGS);

     // ...Lleve a cabo alguna tarea con la base de datos...

     // Finalice la transacción.
     xaResource.end(xid, XAResource.TMSUCCESS);

     // Realice los preparativos para una operación de compromiso.
     xaResource.prepare(xid);
```

```
// Comprometa la transacción.
xaResource.commit(xid, false);

// Cierre la conexión XA cuando haya finalizado. Esto
// cierra el recurso XA implícitamente.
xaConnection.close();
```

Clases de trabajos

Las clases de trabajos de IBM Toolbox para Java (en el paquete access) permiten a un programa Java recuperar y cambiar información de trabajo.

Nota: Toolbox para Java también proporciona clases de recursos que presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación. Las clases de recursos para trabajar con trabajos son RJob, RJobList y RJobLog.

Utilice las clases de trabajos para trabajar con el siguiente tipo de información de trabajo:

- Información de fecha y hora
- Cola de trabajos
- Identificadores de idioma
- Anotaciones de mensajes
- Cola de salida
- Información de impresora

Las clases de trabajos del paquete access son las siguientes:

- Job - recupera y cambia la información de trabajo de iSeries
- JobList - recupera una lista de trabajos de iSeries
- JobLog - representa las anotaciones de trabajo de un iSeries

Ejemplos

Listar los trabajos pertenecientes a un usuario específico y listar los trabajos con información de estado de trabajo.

Visualizar los mensajes que hay en unas anotaciones de trabajo.

Utilizar una antememoria al establecer un valor y al obtener un valor:

```
  try {
      // Crea un objeto AS400.
      AS400 as400 = new AS400("systemName");
      // Construye un objeto Job.
      Job job = new Job(as400,"QDEV002");
      // Obtiene información de trabajo.
      System.out.println("Usuario de este trabajo:" + job.getUser());
      System.out.println("CPU utilizada:" + job.getCPUUsed();
      System.out.println("Fecha del sistema de entrada del trabajo: " +
job.getJobEnterSystemDate());
      // Establece la modalidad de antememoria
      job.setCacheChanges(true);
      // Los cambios se almacenarán en la antememoria.
      job.setRunPriority(66);
      job.setDateFormat("*YMD");
      // Comprometer los cambios. Ello cambiará el valor en el iSeries.
      job.commitChanges();
      // Establecer información de trabajo en el sistema directamente (sin
antememoria).
      job.setCacheChanges(false);
      job.setRunPriority(60);
```

```
} catch (Exception e)
{
    System.out.println(quot;error :" + e)
}
```

Job

La clase Job (en el paquete access) permite a un programa Java recuperar y cambiar información de los trabajos del servidor.

Nota: Toolbox para Java también proporciona clases de recursos que presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación. Las clases de recursos para trabajar con trabajos son RJob, RJobList y RJobLog.

Con la clase Job se puede recuperar y cambiar el siguiente tipo de información de trabajo:

- Colas de trabajo
- Colas de salida
- Anotaciones de mensajes
- Dispositivo de impresora
- Identificador de país o región
- Formato de fecha

La clase Job también ofrece la posibilidad de cambiar un solo valor a la vez o de poner en antememoria varios cambios utilizando el método setCacheChanges(true) y comprometiendo los cambios con el método commitChanges(). Si la puesta en antememoria no está activada, no es necesario realizar un compromiso.

Este ejemplo muestra cómo se establecen valores en la antememoria y se obtienen valores de ella para establecer la prioridad de ejecución con el método setRunPriority() y establecer el formato de fecha con el método setDateFormat().

Clase JobList

Puede utilizar una clase JobList (en el paquete access) para listar trabajos de iSeries.

Nota: Toolbox para Java también proporciona clases de recursos que presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación. Las clases de recursos para trabajar con trabajos son RJob, RJobList y RJobLog.

Con la clase JobList, es posible recuperar:

- Todos los trabajos
- Trabajos por nombre, número de trabajo o usuario

Utilice el método getJobs() para devolver una lista de trabajos de iSeries o el método getLength() para devolver el número de trabajos recuperados con el último método getJobs().

El siguiente ejemplo lista todos los trabajos activos del sistema:

```
    // Cree un objeto AS400. Liste los
    // trabajos que hay en este iSeries.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree el objeto lista de trabajos.
JobList jobList = new JobList(sys);

    // Obtenga la lista de trabajos activos.
Enumeration list = jobList.getJobs();

    // Para cada trabajo activo del sistema,
    // imprima información de trabajo.
while (list.hasMoreElements())
{
       Job j = (Job) list.nextElement();

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

JobLog

La clase JobLog (en el paquete access) recupera los mensajes existentes en las anotaciones de trabajo de un trabajo del servidor efectuando una llamada a getMessages().

Nota: Toolbox para Java también proporciona clases de recursos que presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación. Las clases de recursos para trabajar con trabajos son RJob, RJobList y RJobLog.

El ejemplo siguiente imprime todos los mensajes de las anotaciones de trabajo correspondientes al usuario especificado:

```
          // ...Ya se ha realizado el trabajo de
          // preparación para crear un objeto AS400
          // y un objeto JobList.

          // Obtenga la lista de los trabajos
          // activos en el iSeries
      Enumeration list = jobList.getJobs();

          // Busque en la lista para localizar un
          // trabajo correspondiente al usuario especificado.
      while (list.hasMoreElements())
      {
             Job j = (Job) list.nextElement();

         if (j.getUser().trim().equalsIgnoreCase(userID))
         {
          // Se ha encontrado un trabajo correspondiente
          // al usuario actual. Cree un objeto anotaciones
          // de trabajo para este trabajo.
            JobLog jlog = new JobLog(system,
                                     j.getName(),
                                     j.getUser(),
                                   j.getNumber());

          // Enumere los mensajes de las anotaciones
          // de trabajo y luego imprímalos.
            Enumeration messageList = jlog.getMessages();

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

         }
      }
```

Clases de mensajes

AS400Message

El objeto AS400Message permite al programa Java recuperar un mensaje de iSeries generado en una operación anterior (por ejemplo, en una llamada a mandato). Un programa Java puede recuperar de un objeto mensaje los siguientes datos:

- La biblioteca y el archivo de mensajes de iSeries que contienen el mensaje
- El ID del mensaje
- El tipo de mensaje
- La gravedad del mensaje
- El texto del mensaje
- El texto de ayuda del mensaje

En el ejemplo siguiente se muestra cómo utilizar el objeto AS400Message:

```
    // Cree un objeto llamada a mandato.
CommandCall cmd = new CommandCall(sys, "myCommand");

    // Ejecute el mandato.
cmd.run();

    // Obtenga la lista de mensajes que
    // son el resultado del mandato que
    // acaba de ejecutar.
AS400Message[] messageList = cmd.getMessageList();

    // Itere por la lista
    // para visualizar los mensajes.
for (int i = 0; i < messageList.length; i++)
{
   System.out.println(messageList[i].getText());
}
```

Ejemplos

Los ejemplos que hay a continuación muestran cómo pueden emplearse las listas de mensajes con CommandCall y ProgramCall.

- **Ejemplo:** cómo se utiliza una lista de mensajes con CommandCall
- **Ejemplo:** cómo se utiliza una lista de mensajes con ProgramCall

QueuedMessage

La clase QueuedMessage amplía la clase AS400Message.

Nota: Toolbox para Java también proporciona clases de recursos que presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación. La clase de recurso para trabajar con mensajes en cola es RQueuedMessage.

La clase QueuedMessage accede a información sobre un mensaje de una cola de mensajes de iSeries. Con esta clase,

un programa Java puede recuperar los datos siguientes:

- Información acerca de dónde se originó un mensaje, como por ejemplo, el programa, el nombre de trabajo, el número de trabajo y el usuario
- La cola del mensaje
- La clave del mensaje
- El estado de respuesta del mensaje

El ejemplo siguiente imprime todos los mensajes que hay en la cola de mensajes del usuario actual (el que ha iniciado la sesión):

```
   // La cola de mensajes está en este iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

   // Cree el objeto cola de mensajes.
   // Este objeto representará la cola
   // correspondiente al usuario actual.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

   // Obtenga la lista de mensajes que hay en
   // este momento en la cola de este usuario.
Enumeration e = queue.getMessages();

   // Imprima cada mensaje de la cola.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

MessageFile

La clase MessageFile permite recibir un mensaje de un archivo de mensaje de iSeries. La clase MessageFile devuelve un objeto AS400Message que contiene el mensaje. Mediante la clase MessageFile puede realizar estas tareas:

- Devolver un objeto mensaje que contiene el mensaje
- Devolver un objeto mensaje que contiene texto de sustitución del mensaje

El ejemplo siguiente muestra cómo se recupera e imprime un mensaje:

```
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

La clase MessageQueue permite a un programa Java interactuar con una cola de mensajes de iSeries.

Nota: Toolbox para Java también proporciona clases de recursos que presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación. La clase de recurso para trabajar con colas de mensajes es RMessageQueue.

La clase MessageQueue hace de contenedor para la clase QueuedMessage. El método getMessages(), concretamente, devuelve una lista de objetos QueuedMessage. La clase MessageQueue puede realizar estas tareas:

- Establecer atributos de cola de mensajes
- Obtener información sobre una cola de mensajes
- Recibir mensajes de una cola de mensajes
- Enviar mensajes a una cola de mensajes
- Responder a los mensajes

El ejemplo siguiente lista los mensajes que hay en la cola de mensajes para el usuario actual:

```
   // La cola de mensajes está en este iSeries.
AS400 sys = new AS400(mySystem.myCompany.com);

   // Cree el objeto cola de mensajes.
   // Este objeto representará la cola
   // correspondiente al usuario actual.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

   // Obtenga la lista de mensajes que hay en
   // este momento en la cola de este usuario.
Enumeration e = queue.getMessages();

   // Imprima cada mensaje de la cola.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

NetServer

La clase NetServer representa el servicio NetServer en un servidor iSeries. Los objetos NetServer permiten consultar y modificar el estado y la configuración del servicio NetServer.

Por ejemplo, puede utilizar la clase NetServer para realizar las tareas siguientes:

- Iniciar o detener el NetServer
- Obtener una lista de todos los compartimientos de archivo y compartimientos de impresora actuales
- Obtener una lista de todas las sesiones actuales
- Consultar y cambiar valores de atributo (empleando métodos heredados de ChangeableResource)

Nota: para utilizar la clase NetServer, necesita un perfil de usuario de servidor que tenga la autorización *IOSYSCFG.

La clase NetServer es una extensión de ChangeableResource y Resource, de modo que proporciona un conjunto de "atributos" para representar los diversos valores de NetServer. Puede consultar o cambiar los atributos para acceder a la configuración del NetServer o modificarla. Entre los atributos de NetServer se encuentran los siguientes:

- NAME
- NAME_PENDING
- DOMAIN
- ALLOW_SYSTEM_NAME
- AUTOSTART
- CCSID
- WINS_PRIMARY_ADDRESS

Atributos pendientes

Muchos de los atributos de NetServer son atributos pendientes (por ejemplo, NAME_PENDING). Los atributos pendientes representan los valores de NetServer que entrarán en vigor la próxima vez que se inicie (o que se reinicie) el NetServer en el servidor.

Si tiene un par de atributos relacionados y uno es un atributo pendiente mientras que el otro es un atributo no pendiente:

- El atributo pendiente es de lectura/escritura, por lo que puede modificarlo.
- El atributo no pendiente es de sólo lectura, por lo que puede consultarlo pero no modificarlo.

Otras clases de NetServer

Varias clases de NetServer relacionadas permiten obtener y establecer información detallada sobre conexiones, sesiones, compartimientos de archivo y compartimientos de impresora específicos:

- NetServerConnection - representa una conexión NetServer
- NetServerFileShare - representa un compartimiento de servidor de archivo NetServer
- NetServerPrintShare - representa un compartimiento de servidor de impresora NetServer
- NetServerSession - representa una sesión NetServer
- NetServerShare - representa un compartimiento NetServer

Ejemplo: cómo se utiliza un objeto NetServer para cambiar el nombre del NetServer

```
     // Cree un objeto sistema para representar el servidor iSeries.
     AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWD");
     // Cree un objeto con el que se consultará y modificará el NetServer.
     NetServer nServer = new NetServer(system);
     // Establezca el atributo de nombre pendiente en NEWNAME.
     nServer.setAttributeValue(NetServer.NAME_PENDING, "NEWNAME");
     // Comprometa los cambios. De este modo se envían los cambios al
servidor.
     nServer.commitAttributeChanges();
     // El nombre de NetServer se establecerá en NEWNAME la próxima vez que
se finalice
     // y se inicie NetServer.
```

Clases de permisos

Las clases de permisos permiten obtener y establecer información de autorización sobre objeto. La información de autorización sobre objeto también se llama permiso. La clase Permission representa la autorización que una colección de varios usuarios posee sobre un objeto. La clase UserPermission representa la autorización que un usuario individual posee sobre un objeto específico.

Clase Permission

La clase Permission permite recuperar y cambiar información de autorización sobre objeto. Incluye una colección de diversos usuarios que tienen autorización sobre el objeto. El objeto Permission permite al programa Java poner en antememoria los cambios realizados en la autorización hasta que se llame al método commit(). Una vez llamado el método commit(), todos los cambios realizados hasta ese momento se envían al servidor. Algunas de las funciones proporcionadas por la clase Permission son:

- addAuthorizedUser(): añade un usuario autorizado.
- commit(): compromete en el servidor los cambios realizados en el permiso.
- getAuthorizationList(): devuelve la lista de autorizaciones del objeto.
- getAuthorizedUsers(): devuelve una enumeración de los usuarios autorizados.
- getOwner(): devuelve el nombre del propietario del objeto.
- getSensitivityLevel(): devuelve el nivel de confidencialidad del objeto.
- getType(): devuelve el tipo de autorización sobre objeto (QDLO, QSYS o Raíz).
- getUserPermission(): devuelve el permiso que un determinado usuario tiene sobre el objeto.
- getUserPermissions(): devuelve una enumeración de los permisos que los usuarios tienen sobre el objeto.
- setAuthorizationList(): establece la lista de autorizaciones del objeto.
- setSensitivityLevel(): establece el nivel de confidencialidad del objeto.

Ejemplo

Este ejemplo muestra cómo se crea un permiso y se añade un usuario para que tenga autorización sobre un objeto.

```
// Cree un objeto AS400.
AS400 as400 = new AS400();

// Cree el objeto Permission pasando el AS/400 y el objeto.
Permission myPermission = new Permission(as400, "QSYS.LIB/myLib.LIB");

// Añada un usuario para que tenga autorización sobre el objeto.
myPermission.addAuthorizedUser("User1");
```

Clase UserPermission

La clase UserPermission representa la autorización de un determinado usuario individual.UserPermission tiene tres subclases que manejan la autorización basándose en el tipo de objeto:

Clase UserPermission	Descripción
DLOPermission	Representa la autorización de un usuario sobre los objetos de biblioteca de documentos (DLO), que se almacenan en QDLS.

QSYSPermission	Representa la autorización de un usuario sobre los objetos almacenados en QSYS.LIB y contenidos en el servidor.
RootPermission	Representa la autorización de un usuario sobre los objetos contenidos en la estructura de directorio raíz. Los objetos de RootPermissions son los que no están contenidos en QSYS.LIB ni en QDLS.

La clase UserPermission le permite llevar a cabo estas tareas:

- Determinar si el perfil de usuario es un perfil de grupo
- Devolver el nombre del user perfil
- Indicar si el usuario tiene autorización
- Establecer la autorización de la gestión de lista de autorizaciones

Ejemplo

Este ejemplo muestra cómo se recuperan los usuarios y los grupos que tienen permiso sobre un objeto y cómo se imprimen de uno en uno.

```
      // Cree un objeto sistema.
      AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

      // Represente los permisos sobre un objeto existente en el sistema,
como una biblioteca.
      Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

      // Recupere los diversos usuarios/grupos que tienen establecidos
permisos en ese objeto.
      Enumeration enum = objectInQSYS.getUserPermissions();
      while (enum.hasMoreElements())
      {
      // Imprima los nombres de perfil de usuario/grupo, de uno en uno.
        UserPermission userPerm = (UserPermission)enum.nextElement();
        System.out.println(userPerm.getUserID());
      }
```

Clase DLOPermission

DLOPermission es una subclase de UserPermission. DLOPermission permite visualizar y establecer las autorizaciones (denominadas permisos) que un usuario tiene para un objeto de biblioteca de documentos (DLO).

A cada usuario se le asigna uno de los siguientes valores de autorización:

Valor de autorización	Descripción
*ALL	El usuario puede llevar a cabo todas las operaciones salvo las que están controladas por la gestión de lista de autorizaciones.
*AUTL	Se utiliza la lista de autorizaciones para determinar la autorización sobre el documento.
*CHANGE	El usuario puede realizar cambios y efectuar funciones básicas en el objeto.
*EXCLUDE	El usuario no puede acceder al objeto.
*USE	El usuario posee sobre el objeto autorización operativa, de lectura y de ejecución.

Si desea realizar cambios o determinar cuál es la autorización de un usuario, debe utilizar uno de estos métodos:

- getDataAuthority() permite visualizar el valor de la autorización del usuario
- setDataAuthority() permite establecer el valor de la autorización del usuario

Tras establecer los permisos, es importante que utilice el método commit() de la clase Permissions para enviar los cambios al servidor.

Para obtener más información sobre los permisos y las autorizaciones, consulte el capítulo 5, acerca de la seguridad de recursos de la publicación **iSeries Security Reference**  .

Ejemplo

En este ejemplo se muestra cómo se recuperan e imprimen los permisos DLO, incluidos los perfiles de usuario para cada permiso.

```
      // Cree un objeto sistema.

      AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");
      // Represente los permisos sobre un objeto DLO.
      Permission objectInQDLS = new Permission(sys, "/QDLS/MyFolder");

      // Imprima el nombre de vía del objeto y recupere los correspondientes
permisos.
      System.out.println("Los permisos sobre
"+objectInQDLS.getObjectPath()+" son:");
      Enumeration enum = objectInQDLS.getUserPermissions();
      while (enum.hasMoreElements())
      {
        // Para cada uno de los permisos, imprima el nombre del perfil de
usuario
        // y las autorizaciones que posee dicho usuario sobre el objeto.
        DLOPermission dloPerm = (DLOPermission)enum.nextElement();
        System.out.println(dloPerm.getUserID()+":
"+dloPerm.getDataAuthority());
      }
```

QSYSPermission

QSYSPermission es una subclase de la clase UserPermission. QSYSPermission permite visualizar y establecer el permiso que un usuario posee sobre un objeto de la estructura tradicional de bibliotecas de iSeries o AS/400e almacenado en QSYS.LIB. Puede establecer la autorización sobre un objeto almacenado en QSYS.LIB estableciendo un valor de autorización definido por el sistema o estableciendo las autorizaciones individuales sobre objetos y sobre datos.

La tabla siguiente lista y describe los valores de autorización definidos por el sistema válidos:

Valor de autorización definido por el sistema	Descripción
*ALL	El usuario puede llevar a cabo todas las operaciones salvo las que están controladas por la gestión de lista de autorizaciones.
*AUTL	Se utiliza la lista de autorizaciones para determinar la autorización sobre el documento.
*CHANGE	El usuario puede realizar cambios y efectuar funciones básicas en el objeto.
*EXCLUDE	El usuario no puede acceder al objeto.
*USE	El usuario posee sobre el objeto autorización operativa, de lectura y de ejecución.

Cada valor de autorización definido por el sistema representa en realidad una combinación de las autorizaciones individuales sobre objetos y sobre datos. La tabla siguiente muestra las relaciones de las autorizaciones definidas por el sistema con las autorizaciones individuales sobre objetos y sobre datos:

Autorización definida por el sistema	Autorización sobre objeto					Autorización sobre datos				
	Oper	Gest	Exist	Alter	Ref	Lect	Adic	Actual	Supr	Ejec
Total	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**	**S**
Cambio	**S**	n	n	n	n	**S**	**S**	**S**	**S**	**S**
Exclusión	n	n	n	n	n	n	n	n	n	n
Uso	**S**	n	n	n	n	**S**	n	n	n	**S**
Lista de autorizaciones	Sólo tiene validez con el usuario (*PUBLIC) y una lista de autorizaciones especificada que determine las autorizaciones individuales sobre objetos y sobre datos.									

S indica las autorizaciones que sí pueden asignarse.
n indica las autorizaciones que no pueden asignarse.

Al especificar una autorización definida por el sistema automáticamente se asignan las autorizaciones individuales correspondientes. Del mismo modo, al especificar distintas autorizaciones individuales se cambian los valores de autorización individuales correspondientes. Cuando una combinación de autorizaciones individuales sobre objeto y autorizaciones sobre datos no se correlaciona con un valor de autorización definido por el sistema único, el valor único pasa a ser la autorización "Definida por usuario".

Utilice el método getObjectAuthority() para visualizar la autorización definida por el sistema actual. Utilice el método setObjectAuthority() para establecer la autorización definida por el sistema actual utilizando un solo valor.

Utilice el método set adecuado para habilitar o inhabilitar los valores individuales de autorización sobre objeto:

- setAlter()
- setExistence()
- setManagement()
- setOperational()

- setReference()

Utilice el método set adecuado para habilitar o inhabilitar los valores de autorización sobre datos individuales:

- setAdd()
- setDelete()
- setExecute()
- setRead()
- setUpdate()

Para obtener más información acerca de las distintas autorizaciones, consulte el capítulo 5, acerca de la seguridad de recursos, de la publicación **iSeries Security Reference**  . Para obtener información sobre cómo utilizar los mandatos CL de iSeries para otorgar y editar autorizaciones sobre objetos, consulte los mandatos CL de iSeries Otorgar autorización sobre objeto (GRTOBJAUT) y Editar autorización sobre objeto (EDTOBJAUT).

Ejemplo

Este ejemplo muestra cómo se recuperan e imprimen los permisos correspondientes a un objeto QSYS.

```
      // Cree un objeto sistema.
      AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

      // Represente los permisos sobre un objeto QSYS.
      Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

      // Imprima el nombre de vía del objeto y recupere los correspondientes
permisos.
      System.out.println("Los permisos sobre
"+objectInQSYS.getObjectPath()+" son:");
      Enumeration enum = objectInQSYS.getUserPermissions();
      while (enum.hasMoreElements())
      {
        // Para cada uno de los permisos, imprima el nombre del perfil de
usuario
        // y las autorizaciones que posee dicho usuario sobre el objeto.
        QSYSPermission qsysPerm = (QSYSPermission)enum.nextElement();
        System.out.println(qsysPerm.getUserID()+":
"+qsysPerm.getObjectAuthority());
      }
```

RootPermission

RootPermission es una subclase de la clase UserPermission. La clase RootPermission permite visualizar y establecer los permisos correspondientes a un usuario de un objeto contenido en la estructura del directorio raíz.

Un objeto que esté en la estructura del directorio raíz puede establecer la autorización sobre datos o la autorización sobre objeto. Puede establecer la autorización sobre datos en los valores que se indican en la tabla siguiente. Utilice el método getDataAuthority() para visualizar los valores actuales y el método setDataAuthority() para establecer la autorización sobre datos.

La tabla siguiente lista y describe los valores de autorización sobre datos válidos:

Valor de autorización sobre datos	Descripción
*none	El usuario no posee autorización alguna sobre el objeto.
*RWX	El usuario posee autorización para leer, añadir, actualizar, suprimir y ejecutar.
*RW	El usuario posee autorización para leer, añadir y suprimir.
*RX	El usuario posee autorización para leer y ejecutar.
*WX	El usuario posee autorización para añadir, actualizar, suprimir y ejecutar.
*R	El usuario posee autorización para leer.
*W	El usuario posee autorización para añadir, actualizar y suprimir.
*X	El usuario posee autorización para ejecutar.
*EXCLUDE	El usuario no puede acceder al objeto.
*AUTL	Las autorizaciones de uso público sobre este objeto proceden de la lista de autorizaciones.

La autorización sobre objeto se puede establecer en uno o varios de estos valores: alteración, existencia, gestión o referencia. Puede utilizar los métodos setAlter(), setExistence(), setManagement() o setReference() para activar o desactivar los valores.

Tras establecer la autorización sobre datos o la autorización sobre objeto de un objeto, es importante que utilice el método commit() de la clase Permissions para enviar los cambios al servidor.

Para obtener más información acerca de las distintas autorizaciones, consulte el capítulo 5, acerca de la seguridad de recursos, de la publicación **iSeries Security Reference**  .

Ejemplo

Este ejemplo muestra cómo se recuperan e imprimen los permisos correspondientes a un objeto raíz.

```
      // Cree un objeto sistema.
      AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

      // Represente los permisos sobre un objeto en el sistema de archivos
raíz.
      Permission objectInRoot = new Permission(sys, "/fred");

      // Imprima el nombre de vía del objeto y recupere los correspondientes
permisos.
      System.out.println("Los permisos sobre
"+objectInRoot.getObjectPath()+" son:");
      Enumeration enum = objectInRoot.getUserPermissions();
      while (enum.hasMoreElements())
      {
```

```
        // Para cada uno de los permisos, imprima el nombre del perfil de
usuario
        // y las autorizaciones que posee dicho usuario sobre el objeto.
        RootPermission rootPerm = (RootPermission)enum.nextElement();
        System.out.println(rootPerm.getUserID()+":
"+rootPerm.getDataAuthority());
      }
```

Clases de impresión

Los objetos de impresión son los archivos en spool, las colas de salida, las impresoras, los archivos de impresora, los trabajos transcriptores y los recursos de las funciones avanzadas de impresión (AFP), incluidos los fonts, las definiciones de formulario, los preformatos, las definiciones de página y los segmentos de página. Sólo se puede acceder a los recursos de AFP en OS/400 Versión 3 Release 7 (V3R7) y versiones posteriores. (Si se intenta abrir un objeto AFPResourceList en un sistema que ejecuta una versión anterior a V3R7, se generará una excepción RequestNotSupportedException.)

Las clases de IBM Toolbox para Java de los objetos de impresión están organizadas en una clase base, PrintObject, y en una subclase para cada uno de los seis tipos de objetos de impresión. La clase base contiene los métodos y los atributos que son comunes a todos los objetos de impresión del servidor. Las subclases contienen los métodos y atributos que son específicos para cada subtipo.

Las clases de impresión permiten realizar estas tareas:

- Trabajar con objetos de impresión del servidor:
  - Clase PrintObjectList - utilícela para listar los objetos de impresión del servidor y trabajar con ellos. (Los objetos de impresión son los archivos en spool, las colas de salida, las impresoras, los recursos de las funciones avanzadas de impresión (AFP), los archivos de impresora y los trabajos transcriptores)
  - Clase base PrintObject - utilícela para trabajar con los objetos de impresión
- Recuperar atributos de PrintObject
- Crear nuevos archivos en spool del servidor mediante la clase SpooledFileOutputStream (se utiliza para datos de impresora basados en EBCDIC)
- Generar corrientes de datos de impresora SCS (corriente de caracteres SNA)
- Leer archivos en spool y recursos AFP mediante PrintObjectInputStream
- Leer archivos en spool mediante PrintObjectPageInputStream y PrintObjectTransformedInputStream
- Ver archivos en spool de AFP (funciones avanzadas de impresión) y de SCS (corriente de caracteres SNA)

Ejemplos

- El ejemplo de crear archivo en spool muestra cómo se crea un archivo en spool en un servidor a partir de una corriente de entrada.
- El ejemplo de crear archivo en spool SCS muestra cómo se genera una corriente de datos SCS utilizando la clase SCS3812Writer, y cómo se escribe la corriente en un archivo en spool en el servidor.
- El ejemplo de leer archivo en spool muestra cómo se lee un archivo en spool existente en el servidor.
- El primer ejemplo de lista asíncrona muestra cómo se listan de modo asíncrono todos los archivos en spool de un sistema y cómo se utiliza la interfaz PrintObjectListListener para obtener información de retorno a medida que la lista se va construyendo.
- El segundo ejemplo de lista asíncrona muestra cómo se listan de modo asíncrono todos los archivos en spool de un sistema *sin* utilizar la interfaz PrintObjectListListener.
- El ejemplo de lista síncrona muestra cómo se listan de modo síncrono todos los archivos en spool que hay en un sistema.

Listar objetos de impresión

Puede utilizar la clase PrintObjectList y sus subclases para trabajar con listas de objetos de impresión.Cada subclase dispone de métodos que permiten el filtrado de la lista basándose en los elementos que tienen sentido para un determinado tipo de objeto de impresión. Por ejemplo, SpooledFileList permite filtrar una lista de archivos en spool basándose en el usuario que creó los archivos en spool, en la cola de salida en la que se encuentran los archivos en spool, en el tipo de formulario o en los datos de usuario de los archivos en spool.Únicamente se listan los archivos en spool que coinciden con los criterios de filtrado. De no establecerse ningún filtro, se utiliza un valor por omisión para cada uno de los filtros.

Para recuperar realmente la lista de objetos de impresión del servidor, se emplea el método openSynchronously() o openAsynchronously().El método openSynchronously() no vuelve hasta que se han recuperado del servidor todos los objetos de la lista. El método openAsynchronously() vuelve inmediatamente, y el llamador puede realizar otras tareas en primer plano mientras espera a que se construya la lista. La lista abierta asíncronamente permite asimismo al llamador empezar a mostrar los objetos al usuario a medida que van llegando. El usuario, debido a que puede ver los objetos a medida que van llegando, tiene la impresión de que el tiempo de respuesta es más corto. De hecho, el tiempo de respuesta global puede ser más largo a causa del proceso adicional que se lleva a cabo en cada objeto de la lista.

Si la lista se abre asíncronamente, el llamador puede obtener información de retorno acerca de la construcción de la lista. Diversos métodos, como por ejemplo isCompleted() y size(), indican si ya se ha terminado de construir la lista o devuelven el tamaño actual de la lista.Otros métodos, como waitForListToComplete() y waitForItem(), permiten al llamador esperar a que la lista se complete o a que se recupere un elemento determinado.Además de llamar a estos métodos de PrintObjectList, el llamador puede registrarse en la lista como escuchador. En tal caso, el llamador recibe notificación de los eventos que se producen en la lista. Para registrarse o desregistrarse de los eventos, el llamador utiliza PrintObjectListListener() y a continuación llama a addPrintObjectListListener() para registrarse o a removePrintObjectListListener() para desregistrarse.En la siguiente tabla se muestran los eventos que un objeto PrintObjectList puede comunicar.

Evento de PrintObjectList	Cuándo se entrega el evento
listClosed	Al cerrarse la lista.
listCompleted	Al completarse la lista.
listErrorOccurred	Cuando se lanza alguna excepción mientras se recupera la lista.
listOpened	Al abrirse la lista.
listObjectAdded	Cuando se añade un objeto a la lista.

Tras haber abierto la lista y procesado los objetos que contiene, cierre la lista utilizando el método close().Así se liberan los recursos que se hayan asignado al colector de basura durante la apertura. Después de cerrar una lista, es posible modificar los filtros de la misma, y la lista puede abrirse de nuevo.

Cuando se listan objetos de impresión, los atributos acerca de cada objeto de impresión listado se envían desde el servidor y se almacenan junto con el objeto de impresión. Estos atributos se pueden actualizar mediante el método update() de la clase PrintObject.Los atributos enviados desde el servidor varían en función del tipo de objeto de impresión que se lista. Hay una lista por omisión de atributos para cada tipo de objeto de impresión, que se puede alterar temporalmente utilizando el método setAttributesToRetrieve() de PrintObjectList.En la sección Recuperar atributos de PrintObject encontrará una lista de los atributos soportados por cada tipo de objeto de impresión.

El listado de recursos de AFP sólo está permitido en OS/400 Versión 3 Release 7 y versiones posteriores. Al abrir un objeto AFPResourceList en un sistema cuya versión es anterior a V3R7 se genera una excepción RequestNotSupportedException.

Ejemplos

Ejemplo 1 de lista asíncrona

Ejemplo 2 de lista asíncrona

Ejemplo de lista síncrona

Trabajar con objetos de impresión

PrintObject es una clase abstracta.(Por ser una clase abstracta, no es posible crear una instancia de la clase. En vez de ello, es preciso crear una instancia de una de sus subclases.) Para crear objetos de las subclases, puede seguir cualquiera de estos procedimientos:

- Si conoce el sistema y los atributos identificadores del objeto, construya el objeto explícitamente llamando al constructor público de dicho objeto.
- Puede utilizar una subclase PrintObjectList para construir una lista de los objetos y luego acceder a los objetos individuales mediante la lista.
- Un objeto puede crearse y serle devuelto como resultado de llamar a un método o a un conjunto de métodos. Por ejemplo, el método estático start() de la clase WriterJob devuelve un objeto WriterJob.

Utilice la clase base, PrintObject, y sus subclases para trabajar con los objetos de impresión del servidor:

- OutputQueue
- Printer
- PrinterFile
- SpooledFile
- WriterJob

Recuperar atributos de PrintObject

Para recuperar atributos de un objeto de impresión, puede utilizar el ID de atributo y uno de estos métodos de la clase base PrintObject:

- getIntegerAttribute(int attributeID) permite recuperar un atributo de tipo entero.
- getFloatAttribute(int attributeID) permite recuperar un atributo de tipo coma flotante.
- getStringAttribute(int attributeID) permite recuperar un atributo de tipo serie.

El parámetro attributeID es un entero que identifica el atributo que se ha de recuperar. Todos los ID se definen como constantes públicas en la clase base PrintObject. El archivo PrintAttributes contiene una entrada de cada ID de atributo. La entrada incluye una descripción del atributo y de su tipo (entero, coma flotante o serie). Si desea obtener una lista de cuáles son los atributos que pueden recuperarse mediante estos métodos, seleccione los enlaces siguientes:

- AFPResourceAttrs para recursos de AFP
- OutputQueueAttrs para colas de salida
- PrinterAttrs para impresoras
- PrinterFileAttrs para archivos de impresora
- SpooledFileAttrs para archivos en spool
- WriterJobAttrs para trabajos transcriptores

Para lograr un rendimiento aceptable, estos atributos se copian en el cliente. La copia se realiza al listarse los objetos o bien cuando un objeto se necesite por primera vez, en el caso de que dicho objeto se haya creado implícitamente. Ello evita que el objeto tenga que ir al sistema principal cada vez que la aplicación necesite recuperar un atributo. También hace que sea posible que la instancia del objeto de impresión Java contenga información desfasada acerca del objeto existente en el servidor. El usuario del objeto puede renovar todos los atributos llamando al método update() en el objeto. Además, si la aplicación llama a algún método existente en el objeto que pueda provocar cambios en los atributos del objeto, los atributos se actualizan automáticamente. Por ejemplo, si una cola de salida tiene el atributo de estado RELEASED (getStringAttribute(ATTR_OUTQSTS); devuelve la serie "RELEASED") y se llama al método hold() en la cola de datos, si después se obtiene el atributo de estado, se devolvería el valor HELD.

Método setAttributes

Puede utilizar el método setAttributes para cambiar los atributos de los objetos de archivos en spool y archivo de impresora. Si desea obtener una lista de cuáles son los atributos que se pueden establecer, seleccione los enlaces siguientes:

- PrinterFileAttrs para archivos de impresora
- SpooledFileAttrs para archivos en spool

El método setAttributes tiene un parámetro PrintParameterList, que es una clase utilizada para contener una colección de identificadores de atributos y de sus valores. La lista al principio está vacía, y el llamador puede ir añadiendo atributos a la lista mediante los diversos métodos setParameter().

Clase PrintParameterList

La clase PrintParameterList se puede utilizar para pasar un grupo de atributos a un método que tome como parámetros un número determinado de atributos. Por ejemplo, para enviar un archivo en spool que utilice TCP (LPR), puede emplear el método de SpooledFile, sendTCP(). El objeto PrintParameterList contiene los parámetros necesarios para el mandato de enviar (por ejemplo, el sistema remoto y la cola) más los parámetros opcionales que se deseen (por ejemplo, si hay que suprimir el archivo en spool después del envío). En estos casos, la documentación del método proporciona una lista de los atributos necesarios y de los opcionales. El método setParameter() de PrintParameterList no comprueba qué atributos se establecen ni los valores que se les da. El método setParameter() de PrintParameterList tan solo contiene los valores que se han de pasar al método. En general, los atributos adicionales de

PrintParameterList no se tienen en cuenta, y los valores no permitidos de los atributos utilizados se diagnostican en el servidor.

Atributos de archivo de impresora

Recuperar atributos

Pueden recuperarse los atributos siguientes para un archivo de impresora empleando el método getIntegerAttribute(), getStringAttribute() o getFloatAttribute() adecuado:

- ATTR_ALIGN - Alinear página
- ATTR_BKMGN_ACR - Desplazamiento a través de margen reverso
- ATTR_BKMGN_DWN - Desplazamiento abajo de margen reverso
- ATTR_BACK_OVERLAY - Nombre del sistema de archivos integrado del preformato reverso
- ATTR_BKOVL_DWN - Desplazamiento abajo de preformato reverso
- ATTR_BKOVL_ACR - Desplazamiento a través de preformato reverso
- ATTR_CPI - Caracteres por pulgada
- ATTR_CODEDFNTLIB - Nombre de biblioteca de font codificado
- ATTR_CODEPAGE - Página de códigos
- ATTR_CODEDFNT - Nombre de font codificado
- ATTR_CONTROLCHAR - Carácter de control
- ATTR_CONVERT_LINEDATA - Convertir datos de línea
- ATTR_COPIES - Copias
- ATTR_CORNER_STAPLE - Grapa en esquina
- ATTR_DBCSDATA - Datos DBCS especificados por usuario
- ATTR_DBCSEXTENSN - Caracteres de extensión DBCS
- ATTR_DBCSROTATE - Rotación de caracteres DBCS
- ATTR_DBCSCPI - Caracteres DBCS por pulgada
- ATTR_DBCSSISO - Espaciado DBCS SOSI
- ATTR_DFR_WRITE - Diferir escritura
- ATTR_PAGRTT - Grados de rotación de página
- ATTR_EDGESTITCH_NUMSTAPLES - Número de grapas de ligadura de bordes
- ATTR_EDGESTITCH_REF - Referencia de ligadura de bordes
- ATTR_EDGESTITCH_REFOFF - Desplazamiento de referencia de ligadura de bordes
- ATTR_ENDPAGE - Página final
- ATTR_FILESEP - Separadores de archivo
- ATTR_FOLDREC - Acomodar registros
- ATTR_FONTID - Identificador de font
- ATTR_FORM_DEFINITION - Nombre del sistema de archivos integrado de la definición de formulario
- ATTR_FORMFEED - Alimentación de papel
- ATTR_FORMTYPE - Tipo de formulario
- ATTR_FTMGN_ACR - Desplazamiento a través de margen anverso
- ATTR_FTMGN_DWN - Desplazamiento abajo de margen anverso

- ATTR_FRONT_OVERLAY - Nombre del sistema de archivos integrado del preformato anverso
- ATTR_FTOVL_ACR - Desplazamiento a través de preformato anverso
- ATTR_FTOVL_DWN - Desplazamiento abajo de preformato anverso
- ATTR_CHAR_ID - Juego de caracteres gráficos
- ATTR_JUSTIFY - Alineación de hardware
- ATTR_HOLD - Retener archivo en spool
- ATTR_LPI - Líneas por pulgada
- ATTR_MAXRCDS - Máximo de registros de salida en spool
- ATTR_OUTPTY - Prioridad de salida
- ATTR_OUTPUT_QUEUE - Nombre del sistema de archivos integrado de la cola de salida
- ATTR_OVERFLOW - Número de línea de desbordamiento
- ATTR_PAGE_DEFINITION - Nombre del sistema de archivos integrado de la definición de página
- ATTR_PAGELEN - Longitud de página
- ATTR_MEASMETHOD - Método de medida
- ATTR_PAGEWIDTH - Anchura de página
- ATTR_MULTIUP - Páginas por cara
- ATTR_POINTSIZE - Cuerpo
- ATTR_FIDELITY - Fidelidad de impresión
- ATTR_DUPLEX - Imprimir en ambas caras
- ATTR_PRTQUALITY - Calidad de impresión
- ATTR_PRTTEXT - Texto de impresión
- ATTR_PRINTER - Impresora
- ATTR_PRTDEVTYPE - Tipo de dispositivo de impresora
- ATTR_RPLUNPRT - Sustituir caracteres no imprimibles
- ATTR_RPLCHAR - Carácter de sustitución
- ATTR_SADDLESTITCH_NUMSTAPLES - Número de grapas de cosido por el lomo
- ATTR_SADDLESTITCH_REF - Referencia de cosido por el lomo
- ATTR_SAVE - Guardar archivo en spool
- ATTR_SRCDRWR - Bandeja del papel
- ATTR_SPOOL - Poner los datos en spool
- ATTR_SCHEDULE - Planificación de salida en spool
- ATTR_STARTPAGE - Página inicial
- ATTR_DESCRIPTION - Descripción de texto
- ATTR_UNITOFMEAS - Unidad de medida
- ATTR_USERDATA - Datos de usuario
- ATTR_USRDEFDATA - Datos definidos por usuario
- ATTR_USRDEFOPT - Opciones definidas por el usuario
- ATTR_USER_DEFINED_OBJECT - Nombre del sistema de archivos integrado del objeto definido por el

usuario

Establecer atributos

Pueden establecerse los atributos siguientes para un archivo de impresora empleando el método setAttributes():

- ATTR_ALIGN - Alinear página
- ATTR_BKMGN_ACR - Desplazamiento a través de margen reverso
- ATTR_BKMGN_DWN - Desplazamiento abajo de margen reverso
- ATTR_BACK_OVERLAY - Nombre del sistema de archivos integrado del preformato reverso
- ATTR_BKOVL_DWN - Desplazamiento abajo de preformato reverso
- ATTR_BKOVL_ACR - Desplazamiento a través de preformato reverso
- ATTR_CPI - Caracteres por pulgada
- ATTR_CODEDFNTLIB - Nombre de biblioteca de font codificado
- ATTR_CODEPAGE - Página de códigos
- ATTR_CODEDFNT - Nombre de font codificado
- ATTR_CONTROLCHAR - Carácter de control
- ATTR_CONVERT_LINEDATA - Convertir datos de línea
- ATTR_COPIES - Copias
- ATTR_CORNER_STAPLE - Grapa en esquina
- ATTR_DBCSDATA - Datos DBCS especificados por usuario
- ATTR_DBCSEXTENSN - Caracteres de extensión DBCS
- ATTR_DBCSROTATE - Rotación de caracteres DBCS
- ATTR_DBCSCPI - Caracteres DBCS por pulgada
- ATTR_DBCSSISO - Espaciado DBCS SOSI
- ATTR_DFR_WRITE - Diferir escritura
- ATTR_PAGRTT - Grados de rotación de página
- ATTR_EDGESTITCH_NUMSTAPLES - Número de grapas de ligadura de bordes
- ATTR_EDGESTITCH_REF - Referencia de ligadura de bordes
- ATTR_EDGESTITCH_REFOFF - Desplazamiento de referencia de ligadura de bordes
- ATTR_ENDPAGE - Página final
- ATTR_FILESEP - Separadores de archivo
- ATTR_FOLDREC - Acomodar registros
- ATTR_FONTID - Identificador de font
- ATTR_FORM_DEFINITION - Nombre del sistema de archivos integrado de la definición de formulario
- ATTR_FORMFEED - Alimentación de papel
- ATTR_FORMTYPE - Tipo de formulario
- ATTR_FTMGN_ACR - Desplazamiento a través de margen anverso
- ATTR_FTMGN_DWN - Desplazamiento abajo de margen anverso
- ATTR_FRONT_OVERLAY - Nombre del sistema de archivos integrado del preformato anverso

- ATTR_FTOVL_ACR - Desplazamiento a través de preformato anverso
- ATTR_FTOVL_DWN - Desplazamiento abajo de preformato anverso
- ATTR_CHAR_ID - Juego de caracteres gráficos
- ATTR_JUSTIFY - Alineación de hardware
- ATTR_HOLD - Retener archivo en spool
- ATTR_LPI - Líneas por pulgada
- ATTR_MAXRCDS - Máximo de registros de salida en spool
- ATTR_OUTPTY - Prioridad de salida
- ATTR_OUTPUT_QUEUE - Nombre del sistema de archivos integrado de la cola de salida
- ATTR_OVERFLOW - Número de línea de desbordamiento
- ATTR_PAGE_DEFINITION - Nombre del sistema de archivos integrado de la definición de página
- ATTR_PAGELEN - Longitud de página
- ATTR_MEASMETHOD - Método de medida
- ATTR_PAGEWIDTH - Anchura de página
- ATTR_MULTIUP - Páginas por cara
- ATTR_POINTSIZE - Cuerpo
- ATTR_FIDELITY - Fidelidad de impresión
- ATTR_DUPLEX - Imprimir en ambas caras
- ATTR_PRTQUALITY - Calidad de impresión
- ATTR_PRTTEXT - Texto de impresión
- ATTR_PRINTER - Impresora
- ATTR_PRTDEVTYPE - Tipo de dispositivo de impresora
- ATTR_RPLUNPRT - Sustituir caracteres no imprimibles
- ATTR_RPLCHAR - Carácter de sustitución
- ATTR_SADDLESTITCH_NUMSTAPLES - Número de grapas de cosido por el lomo
- ATTR_SADDLESTITCH_REF - Referencia de cosido por el lomo
- ATTR_SAVE - Guardar archivo en spool
- ATTR_SRCDRWR - Bandeja del papel
- ATTR_SPOOL - Poner los datos en spool
- ATTR_SCHEDULE - Planificación de salida en spool
- ATTR_STARTPAGE - Página inicial
- ATTR_DESCRIPTION - Descripción de texto
- ATTR_UNITOFMEAS - Unidad de medida
- ATTR_USERDATA - Datos de usuario
- ATTR_USRDEFDATA - Datos definidos por usuario
- ATTR_USRDEFOPT - Opciones definidas por el usuario
- ATTR_USER_DEFINED_OBJECT - Nombre del sistema de archivos integrado del objeto definido por el usuario

Atributos de archivo en spool

Recuperar atributos

Pueden recuperarse los atributos siguientes para un archivo en spool empleando el método getIntegerAttribute(), getStringAttribute() o getFloatAttribute() adecuado:

- ATTR_AFP - Funciones avanzadas de impresión
- ATTR_ALIGN - Alinear página
- ATTR_BKMGN_ACR - Desplazamiento a través de preformato reverso
- ATTR_BKMGN_DWN - Desplazamiento abajo de preformato reverso
- ATTR_BACK_OVERLAY - Nombre del sistema de archivos integrado del preformato reverso
- ATTR_BKOVL_DWN - Desplazamiento abajo de preformato reverso
- ATTR_BKOVL_ACR - Desplazamiento a través de preformato reverso
- ATTR_CPI - Caracteres por pulgada
- ATTR_CODEDFNTLIB - Nombre de biblioteca de font codificado
- ATTR_CODEDFNT - Nombre de font codificado
- ATTR_CODEPAGE - Página de códigos
- ATTR_CONTROLCHAR - Carácter de control
- ATTR_COPIES - Copias
- ATTR_COPIESLEFT - Copias dejadas para producir
- ATTR_CORNER_STAPLE - Grapa en esquina
- ATTR_CURPAGE - Página actual
- ATTR_DATE - Fecha de creación del objeto
- ATTR_DATE_WTR_BEGAN_FILE - Fecha en que el transcriptor empezó a procesar el archivo en spool
- ATTR_DATE_WTR_CMPL_FILE - Fecha en que el transcriptor terminó de procesar el archivo en spool
- ATTR_DBCSDATA - Datos DBCS especificados por usuario
- ATTR_DBCSEXTENSN - Caracteres de extensión DBCS
- ATTR_DBCSROTATE - Rotación de caracteres DBCS
- ATTR_DBCSCPI - Caracteres DBCS por pulgada
- ATTR_DBCSSISO - Espaciado DBCS SOSI
- ATTR_PAGRTT - Grados de rotación de página
- ATTR_EDGESTITCH_NUMSTAPLES - Número de grapas de ligadura de bordes
- ATTR_EDGESTITCH_REF - Referencia de ligadura de bordes
- ATTR_EDGESTITCH_REFOFF - Desplazamiento de referencia de ligadura de bordes
- ATTR_ENDPAGE - Página final
- ATTR_FILESEP - Separadores de archivo
- ATTR_FOLDREC - Acomodar registros

- ATTR_FONTID - Identificador de font
- ATTR_FORM_DEFINITION - Nombre del sistema de archivos integrado de la definición de formulario
- ATTR_FORMFEED - Alimentación de papel
- ATTR_FORMTYPE - Tipo de formulario
- ATTR_FTMGN_ACR - Desplazamiento a través de margen anverso
- ATTR_FTMGN_DWN - Desplazamiento abajo de margen anverso
- ATTR_FRONTSIDE_OVERLAY - Nombre del sistema de archivos integrado del preformato anverso
- ATTR_FTOVL_ACR - Desplazamiento a través de preformato anverso
- ATTR_FTOVL_DWN - Desplazamiento abajo de preformato anverso
- ATTR_CHAR_ID - Juego de caracteres gráficos
- ATTR_JUSTIFY - Alineación de hardware
- ATTR_HOLD - Retener archivo en spool
- ATTR_IPP_ATTR_CHARSET - Atributos IPP - juego de caracteres
- ATTR_IPP_JOB_ID - ID de trabajo IPP
- ATTR_IPP_JOB_NAME - Nombre de trabajo IPP
- ATTR_IPP_JOB_NAME_NL - NL de nombre de trabajo IPP
- ATTR_IPP_JOB_ORIGUSER - Nombre de usuario emisor de trabajo IPP
- ATTR_IPP_JOB_ORIGUSER_NL - NL de nombre de usuario emisor de trabajo IPP
- ATTR_IPP_PRINTER_NAME - Nombre de impresora IPP
- ATTR_JOBNAME - Nombre de trabajo
- ATTR_JOBNUMBER - Número de trabajo
- ATTR_JOBUSER - Usuario de trabajo
- » ATTR_JOB_SYSTEM - Sistema de trabajo «
- ATTR_LASTPAGE - Última página impresa
- ATTR_LINESPACING - Interlineado
- ATTR_LPI - Líneas por pulgada
- ATTR_MAXRCDS - Máximo de registros de salida en spool
- ATTR_PAGELEN - Longitud de página
- ATTR_PAGEWIDTH - Anchura de página
- ATTR_MEASMETHOD - Método de medida
- ATTR_NETWORK - Identificador de red
- ATTR_NUMBYTES - Número de bytes de lectura/escritura
- ATTR_OUTPUTBIN - Bandeja de salida
- ATTR_OUTPTY - Prioridad de salida
- ATTR_OUTPUT_QUEUE - Nombre del sistema de archivos integrado de la cola de salida
- ATTR_OVERFLOW - Número de línea de desbordamiento
- ATTR_MULTIUP - Páginas por cara
- ATTR_POINTSIZE - Cuerpo

- ATTR_FIDELITY - Fidelidad de impresión
- ATTR_DUPLEX - Imprimir en ambas caras
- ATTR_PRTQUALITY - Calidad de impresión
- ATTR_PRTTEXT - Texto de impresión
- ATTR_PRINTER - Impresora
- ATTR_PRTASSIGNED - Impresora asignada
- ATTR_PRTDEVTYPE - Tipo de dispositivo de impresora
- ATTR_PRINTER_FILE - Nombre del sistema de archivos integrado del archivo de impresora
- ATTR_RECLENGTH - Longitud de registro
- ATTR_REDUCE - Reducir salida
- ATTR_RPLUNPRT - Sustituir caracteres no imprimibles
- ATTR_RPLCHAR - Carácter de sustitución
- ATTR_RESTART - Reiniciar impresión
- ATTR_SADDLESTITCH_NUMSTAPLES - Número de grapas de cosido por el lomo
- ATTR_SADDLESTITCH_REF - Referencia de cosido por el lomo
- ATTR_SAVE - Guardar archivo en spool
- ATTR_SRCDRWR - Bandeja del papel
- ATTR_SPOOLFILE - Nombre de archivo en spool
- ATTR_SPLFNUM - Número de archivo en spool
- ATTR_SPLFSTATUS - Estado de archivo en spool
- ATTR_SCHEDULE - Planificación de salida en spool
- ATTR_STARTPAGE - Página inicial
- ATTR_SYSTEM - Sistema en el que se ha creado
- ATTR_TIME - Hora de creación del objeto
- ATTR_TIME_WTR_BEGAN_FILE - Hora en que el transcriptor empezó a procesar el archivo en spool
- ATTR_TIME_WTR_CMPL_FILE - Hora en que el transcriptor terminó de procesar el archivo en spool
- ATTR_PAGES - Total de páginas
- ATTR_UNITOFMEAS - Unidad de medida
- ATTR_USERCMT - Comentario de usuario
- ATTR_USERDATA - Datos de usuario
- ATTR_USRDEFDATA - Datos definidos por usuario
- ATTR_USRDEFFILE - Archivo definido por usuario
- ATTR_USRDEFOPT - Opciones definidas por el usuario
- ATTR_USER_DEFINED_OBJECT - Nombre del sistema de archivos integrado del objeto definido por el usuario

---

Establecer atributos

Pueden establecerse los atributos siguientes para un archivo en spool empleando el método setAttributes():

- ATTR_ALIGN - Alinear página
- ATTR_BACK_OVERLAY - Nombre del sistema de archivos integrado del preformato reverso
- ATTR_BKOVL_DWN - Desplazamiento abajo de preformato reverso
- ATTR_BKOVL_ACR - Desplazamiento a través de preformato reverso
- ATTR_COPIES - Copias
- ATTR_ENDPAGE - Página final
- ATTR_FILESEP - Separadores de archivo
- ATTR_FORM_DEFINITION - Nombre del sistema de archivos integrado de la definición de formulario
- ATTR_FORMFEED - Alimentación de papel
- ATTR_FORMTYPE - Tipo de formulario
- ATTR_FRONTSIDE_OVERLAY - Nombre del sistema de archivos integrado del preformato anverso
- ATTR_FTOVL_ACR - Desplazamiento a través de preformato anverso
- ATTR_FTOVL_DWN - Desplazamiento abajo de preformato anverso
- ATTR_OUTPTY - Prioridad de salida
- ATTR_OUTPUT_QUEUE - Nombre del sistema de archivos integrado de la cola de salida
- ATTR_MULTIUP - Páginas por cara
- ATTR_FIDELITY - Fidelidad de impresión
- ATTR_DUPLEX - Imprimir en ambas caras
- ATTR_PRTQUALITY - Calidad de impresión
- ATTR_PRTSEQUENCE - Secuencia de impresión
- ATTR_PRINTER - Impresora
- ATTR_RESTART - Reiniciar impresión
- ATTR_SAVE - Guardar archivo en spool
- ATTR_SCHEDULE - Planificación de salida en spool
- ATTR_STARTPAGE - Página inicial
- ATTR_USERDATA - Datos de usuario
- ATTR_USRDEFOPT - Opciones definidas por el usuario
- ATTR_USER_DEFINED_OBJECT - Nombre del sistema de archivos integrado del objeto definido por el usuario

---

Crear archivos en spool nuevos

Puede utilizar la clase SpooledFileOutputStream para crear nuevos archivos en spool del servidor. La clase se deriva de la clase java.io.OutputStream de JDK estándar; una vez construida, puede emplearse dondequiera que se utilice una corriente de salida (OutputStream).

Al crear un objeto SpooledFileOutputStream nuevo, el llamador puede especificar los elementos siguientes:

- El archivo de impresora que se ha de usar
- La cola de salida en la que hay que poner el archivo en spool
- Un objeto PrintParameterList que puede contener parámetros para alterar temporalmente los campos del archivo de impresora

Todos estos parámetros son opcionales (el llamador puede pasarlos todos o no pasar ninguno de ellos). Si no se especifica un archivo de impresora, el servidor de impresión de red utiliza el archivo QPNPSPRTF, que es el archivo de impresora por omisión para la impresión de red. También está el parámetro cola de salida, porque es práctico; este parámetro se puede especificar en PrintParameterList. Si el parámetro cola de salida se especifica en los dos lugares, el campo de PrintParameterList altera temporalmente el parámetro cola de salida. En la documentación del constructor de SpooledFileOutputStream encontrará una lista completa de qué atributos se pueden establecer en PrintParameterList para crear nuevos archivos en spool.

Utilice uno de los métodos write() para escribir datos en el archivo en spool. El objeto SpooledFileOutputStream pone los datos en el almacenamiento intermedio y los envía cuando se cierra la corriente de salida o al llenarse el almacenamiento intermedio. La puesta en el almacenamiento intermedio se realiza por dos razones:

- Permite a la función que determina automáticamente el tipo de datos (consulte Tipos de corriente de datos en archivos en spool) analizar un almacenamiento intermedio de datos lleno para determinar el tipo de datos.
- Agiliza el funcionamiento de la corriente de salida ya que no todas las peticiones de escritura se comunican al servidor.

Utilice el método flush() para forzar que los datos se escriban en el servidor.

Cuando el llamador ha terminado de escribir los datos en el nuevo archivo en spool, se llama al método close() para cerrar el archivo en spool. Una vez cerrado, no se pueden escribir más datos en él. Al llamar al método getSpooledFile() una vez cerrado el archivo en spool, el llamador puede obtener una referencia a un objeto SpooledFile que representa el archivo en spool.

Tipos de corriente de datos en archivos en spool

El atributo "Tipo de datos de impresora" del archivo en spool permite establecer el tipo de datos que se ha de poner en el archivo en spool. Si el llamador no especifica un tipo de datos de impresora, el valor por omisión es utilizar la determinación automática de tipo de datos. Este método examina los mil primeros bytes de los datos de archivo en spool, determina si se ajustan a las arquitecturas de corriente de datos SCS (corriente de caracteres SNA) o AFPDS (corriente de datos de las funciones avanzadas de impresión), y luego establece el atributo de manera adecuada. Si los bytes de los datos de archivo en spool no coinciden con ninguna de estas arquitecturas, los datos se marcan como de tipo *USERASCII. La determinación automática de tipo de datos funciona la mayor parte del tiempo. En general, el llamador debe utilizar esta función a menos que se trate de un caso específico en el que no funcione la determinación automática de tipo de datos. En tales casos, el llamador puede hacer que el atributo "Tipo de datos de impresora" se establezca en un valor específico (por ejemplo, *SCS). El llamador, si desea utilizar los datos de impresora que están en el archivo de impresora, debe utilizar el valor especial *PRTF. El llamador, si altera temporalmente el tipo de datos por omisión al crear un archivo en spool, debe tener la precaución de asegurarse de que los datos que se ponen en el archivo en spool coinciden con el atributo de tipo de datos. Poner datos no SCS en un archivo en spool marcado para recibir datos SCS hace que el sistema principal desencadene un mensaje de error, y provoca la pérdida del archivo en spool.

En general, este atributo puede tener tres valores:

- ***SCS** - una corriente de datos de impresora basada en texto y EBCDIC.

- ***AFPDS** (corriente de datos de funciones avanzadas de presentación) - otra corriente de datos soportada en el servidor. *AFPDS puede contener texto, imagen y gráficos, y puede usar recursos externos, como preformatos de página e imágenes externas en segmentos de página.
- ***USERASCII** - cualesquiera datos de impresora que no sean SCS ni AFPDS manejados por el servidor mediante el paso a través. Postscript y HP-PCL son ejemplos de corrientes de datos que se pondrían en un archivo en spool de tipo *USERASCII.

Ejemplos

Ejemplo de crear archivo en spool

Ejemplo de crear archivo en spool SCS

Generar una corriente de datos SCS

Para generar archivos en spool que se impriman en determinadas impresoras conectadas al servidor, puede ser necesario crear una corriente de datos SCS (corriente de caracteres SNA). (SCS es una corriente de datos EBCDIC basada en texto, que puede imprimirse en impresoras SCS, en impresoras IPDS o en impresoras de PC.) La impresión de SCS se puede realizar convirtiendo dicha corriente de datos mediante un emulador o con la transformación de impresión en sistema principal en el servidor.

Puede utilizar las clases de transcriptor (writer) SCS para generar una corriente de datos SCS de este tipo. Las clases de transcriptor SCS convierten los caracteres Unicode Java y las opciones de formato en una corriente de datos SCS. Hay cinco clases de transcriptor SCS que generan los diversos niveles de corrientes de datos SCS. El llamador debe elegir el transcriptor que corresponde al destino de impresora final en el que va a imprimir el llamador o el usuario final.

Para generar una corriente de datos de impresora SCS, utilice las clases de transcriptor SCS siguientes:

Clase de transcriptor SCS	Descripción
SCS5256Writer	La clase de transcriptor SCS más simple de todas. Soporta texto, retorno de carro, salto de línea, nueva línea, salto de página, orientación horizontal y vertical absoluta, orientación horizontal y vertical relativa y establecer formato vertical.
SCS5224Writer	Amplía la clase de transcriptor 5256 y añade métodos para establecer los caracteres por pulgada (CPI) y las líneas por pulgada (LPI).
SCS5219Writer	Amplía la clase de transcriptor 5224 y añade soporte para margen izquierdo, subrayado, tipo de papel (papel o sobre), tamaño de papel, calidad de impresión, página de códigos, juego de caracteres, número de bandeja alimentadora y número de bandeja destino.
SCS5553Writer	Amplía la clase de transcriptor 5219 y añade soporte para rotación de caracteres, líneas de cuadrícula y ajuste de fonts. 5553 es una corriente de datos DBCS (juego de caracteres de doble byte).
SCS3812Writer	Amplía la clase de transcriptor 5219 y añade soporte para negrita, dúplex, orientación de texto y fonts.

Para construir un transcriptor SCS, el llamador necesita una corriente de salida y, opcionalmente, una codificación. La corriente de datos se escribe en la corriente de salida. Para crear un archivo en spool SCS, el llamador construye primero un objeto SpooledFileOutputStream, y luego lo utiliza para construir un objeto transcriptor SCS. El parámetro de codificación proporciona un identificador de juego de caracteres codificados (CCSID) EBCDIC al que se han de convertir los datos.

Una vez construido el transcriptor, los métodos write() permiten enviar texto a la salida. Los métodos carriageReturn(), lineFeed() y newLine() permiten situar el cursor de escritura en la página. El método endPage() permite finalizar la página actual e iniciar una página nueva.

Tras escribir todos los datos, utilice el método close() para finalizar la corriente de datos y cerrar la corriente de salida.

Leer archivos en spool y recursos AFP

Puede utilizar la clase PrintObjectInputStream para leer el contenido sin procesar de un archivo en spool o un recurso de AFP (funciones avanzadas de impresión) del servidor. Esta clase amplía la clase java.io.InputStream de JDK estándar para que pueda emplearse dondequiera que se utilice una corriente de entrada (InputStream).

Puede obtener un objeto PrintObjectInputStream llamando al método getInputStream() en una instancia de la clase SpooledFile o al método getInputStream() en una instancia de la clase AFPResource. La obtención de una corriente de entrada para un archivo en spool está soportada en las versiones V3R2 (Versión 3 Release 2), V3R7 y posteriores de OS/400. La obtención de corrientes de entrada para recursos AFP está soportada en una versión que sea igual o posterior a V3R7.

Utilice uno de los métodos read() para leer en la corriente de entrada. Todos estos métodos devuelven el número de bytes leídos realmente, o bien -1 en el caso de que no se haya leído ningún byte y de que se haya llegado al final del archivo.

Utilice el método available() de PrintObjectInputStream para devolver el número total de bytes del archivo en spool o del recurso AFP. La clase PrintObjectInputStream da soporte a marcar la corriente de entrada, por lo que PrintObjectInputStream siempre devuelve true desde el método markSupported(). El llamador puede utilizar los métodos mark() y reset() para hacer que la posición de lectura actual retroceda en la corriente de entrada. Utilice el método skip() para hacer que la posición de lectura avance en la corriente de entrada sin leer los datos.

Ejemplo

Ejemplo de leer un archivo en spool

Leer archivos en spool mediante PrintObjectPageInputStream y PrintObjectTransformedInputStream

Puede utilizar la clase PrintObjectPageInputStream para leer los datos de un archivo en spool AFP y SCS del servidor, de página en página.

Puede obtener un objeto PrintObjectPageInputStream con el método getPageInputStream().

Utilice uno de los métodos read() para leer en la corriente de entrada. Todos ellos devuelven el número de bytes leídos realmente, o bien -1 en el caso de que no se haya leído ningún byte y de que se haya llegado al final de la página.

Utilice el método available() de PrintObjectPageInputStream para devolver el número total de bytes de la página actual. La clase PrintObjectPageInputStream da soporte a marcar la corriente de entrada, por lo que PrintObjectPageInputStream siempre devuelve true desde el método markSupported(). El llamador puede utilizar los métodos mark() y reset() para hacer que la posición de lectura actual retroceda en la corriente de entrada y así lecturas posteriores vuelvan a leer los mismos bytes. El llamador puede utilizar el método skip() para hacer que la posición de lectura avance en la corriente de entrada sin leer los datos.

Sin embargo, si desea transformar toda una corriente de datos de archivo en spool, utilice la clase PrintObjectTransformedInputStream.

Licencia de producto

La clase ProductLicense permite solicitar las licencias de productos instalados en el iSeries. Para que sea compatible con otros usuarios de licencias de iSeries, la clase funciona mediante el soporte para licencias de productos de iSeries al solicitar o liberar una licencia.

La clase no aplica la política de licencias sino que devuelve suficiente información para que la aplicación pueda implementarla. Cuando se solicita una licencia, la clase ProductLicense devuelve el estado de la petición (licencia concedida o denegada). Si se deniega la petición, la aplicación debe inhabilitar el comportamiento que requería la licencia ya que IBM Toolbox para Java no sabe qué función debe inhabilitar.

Utilice la clase ProductLicense con el soporte para licencias de iSeries con objeto de implementar la licencia de la aplicación:

- El lado servidor de la aplicación registra el producto y los términos de la licencia con el soporte para licencias de iSeries.
- El lado cliente de la aplicación utiliza el objeto ProductLicense para solicitar y liberar licencias.

Ejemplo: caso práctico de ProductLicense

Imagine, por ejemplo, que un cliente ha adquirido 15 licencias de uso simultáneas de su producto. El uso simultáneo significa que 15 usuarios pueden utilizar el producto a la vez, pero no tienen que ser necesariamente 15 usuarios específicos. Pueden ser 15 usuarios cualesquiera de la organización. Esta información se registra con el soporte para licencias de iSeries. A medida que los usuarios se conectan, la aplicación utiliza la clase ProductLicense para solicitar una licencia.

- Si el número de usuarios simultáneos es inferior a 15, la petición es satisfactoria y se ejecuta la aplicación.
- Cuando se conecta el usuario 16, la petición ProductLicense falla. La aplicación visualiza un mensaje de error y se termina.

Cuando un usuario deja de ejecutar la aplicación, la aplicación libera la licencia mediante la clase ProductLicense. En ese momento la licencia ya está disponible para que otro usuario la utilice.

Para obtener más información y un ejemplo de código, consulte el javadoc de ProductLicense.

Clase ProgramCall

La clase ProgramCall permite al programa Java llamar a un programa de iSeries. Puede utilizar la clase ProgramParameter para especificar parámetros de entrada, de salida y de entrada/salida. Si el programa se ejecuta, los parámetros de salida y de entrada/salida contienen los datos devueltos por el programa de iSeries. Si el programa de iSeries no puede ejecutarse satisfactoriamente, el programa Java puede recuperar los mensajes de iSeries que se produzcan en forma de una lista de objetos AS400Message.

Los parámetros obligatorios son:

- El programa y los parámetros que han de ejecutarse.
- El objeto AS400 que representa el sistema iSeries que tiene el programa.

El nombre de programa y la lista de parámetros se pueden establecer en el constructor, mediante el método setProgram() o se pueden establecer en el método run(). El método run() llama al programa.

La clase de objeto ProgramCall hace que el objeto AS400 se conecte al iSeries.

El siguiente ejemplo muestra cómo se utiliza la clase ProgramCall:

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree un objeto programa. En este ejemplo,
    // se establece que el programa se ejecute más adelante.
ProgramCall pgm = new ProgramCall(sys);

    // Establezca el nombre del programa.
    // Debido a que el programa no toma ningún
    // parámetro, pase null para el argumento
    // de ProgramParameter[].
pgm.setProgram(QSYSObjectPathName.toPath("MYLIB",
                                         "MYPROG",
                                         "PGM"));

    // Ejecute el programa. En este ejemplo, el programa
    // no tiene ningún parámetro. Si no se ejecuta, la anomalía
    // se devuelve como conjunto de mensajes en la
    // lista de mensajes.
if (pgm.run() != true)
{
    // Si llega a este punto, es que el programa
    // no ha podido ejecutarse. Obtenga la lista de
    // mensajes para determinar por qué el
    // programa no se ejecutó.
   AS400Message[] messageList = pgm.getMessageList();

    // ...Procese la lista de mensajes.
}

    // Desconecte, puesto que ya ha terminado
    // de ejecutar los programas.
sys.disconnectService(AS400.COMMAND);
```

El objeto ProgramCall requiere el nombre de vía de acceso del sistema de archivos integrado del programa.

La utilización de la clase ProgramCall hace que el objeto AS400 se conecte al iSeries. En Gestión de conexiones

encontrará información acerca de cómo se gestionan las conexiones.

El comportamiento por omisión consiste en el caso de los programas de iSeries en que la ejecución se lleve a cabo en un trabajo servidor aparte, aun cuando el programa Java y el programa de iSeries estén en el mismo servidor. Puede alterar temporalmente el comportamiento por omisión y hacer que el programa de iSeries se ejecute en el trabajo Java mediante el método setThreadSafe().

Utilización de los objetos ProgramParameter

Los objetos ProgramParameter permiten pasar datos de parámetro entre el programa Java y el programa de iSeries. Establezca los datos de entrada con el método setInputData(). Tras la ejecución del programa, recupere los datos de salida con el método getOutputData(). Cada parámetro es una matriz de bytes. El programa Java debe convertir la matriz de bytes entre el formato Java y el formato de iSeries. Las clases de conversión de datos proporcionan métodos para convertir los datos. Los parámetros se añaden al objeto ProgramCall en forma de lista.

El ejemplo siguiente muestra cómo se utiliza el objeto ProgramParameter para pasar datos de parámetro.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // En este ejemplo, el programa tiene dos
    // parámetros. Cree una lista que contenga
    // dichos parámetros.
ProgramParameter[] parmList = new ProgramParameter[2];

    // El primero es un parámetro
    // de entrada.
byte[] key = {1, 2, 3};
parmList[0] = new ProgramParameter(key);

    // El segundo es un parámetro
    // de salida. Se devuelve un
    // número de cuatro bytes.
parmList[1] = new ProgramParameter(4);

    // Cree un objeto programa
    // especificando el nombre del programa
    // y la lista de parámetros.
ProgramCall pgm = new ProgramCall(sys,
                                  "/QSYS.LIB/MYLIB.LIB/MYPROG.PGM",
                                  parmList);

    // Ejecute el programa.
if (pgm.run() != true)
{

    // Si el iSeries no puede ejecutar el
    // programa, vea la lista de mensajes para
    // averiguar por qué no se ha ejecutado.
   AS400Message[] messageList = pgm.getMessageList();

}
     else
{
    // En caso contrario, el programa se ha ejecutado.
    // Procese el segundo parámetro, que contiene
    // los datos devueltos.
```

```
    // Cree un conversor para este
    // tipo de datos iSeries.
   AS400Bin4 bin4Converter = new AS400Bin4();

    // Realice la conversión desde el tipo iSeries al
    // objeto Java. El número empieza al principio
    // del almacenamiento intermedio.
   byte[] data = parmList[1].getOutputData();
   int i = bin4Converter.toInt(data);
}

    // Desconecte, puesto que ya ha terminado
    // de ejecutar los programas.
sys.disconnectService(AS400.COMMAND);
```

Clase QSYSObjectPathName

Puede utilizar la clase QSYSObjectPathName para representar un objeto en el sistema de archivos integrado. Esta clase permite construir un nombre de sistema de archivos integrado o analizar en sus componentes un nombre de sistema de archivos integrado.

Varias de las clases de IBM Toolbox para Java necesitan un nombre de vía de acceso del sistema de archivos integrado para poderse utilizar. El objeto QSYSObjectPathName permite construir el nombre.

Los ejemplos que hay a continuación muestran cómo se utiliza la clase QSYSObjectPathName:

Ejemplo 1: el objeto ProgramCall requiere el nombre de sistema de archivos integrado del programa del servidor al que se debe llamar. Se utiliza un objeto QSYSObjectPathName para construir el nombre. Para llamar al programa PRINT_IT de la biblioteca REPORTS utilizando un objeto QSYSObjectPathName:

```
  // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree un objeto llamada a programa.
ProgramCall pgm = new ProgramCall(sys);

    // Cree un objeto nombre de vía de acceso que
    // represente el programa PRINT_IT de la
    // biblioteca REPORTS.
QSYSObjectPathName pgmName = new QSYSObjectPathName("REPORTS",
                                        "PRINT_IT",
                                                "PGM");

    // Utilice el objeto nombre de vía para
    // establecer el nombre en el objeto
    // llamada a programa.
pgm.setProgram(pgmName.getPath());

    // ...Ejecute el programa, procese
    // los resultados.
```

Ejemplo 2: si el nombre del objeto AS400 sólo se utiliza una vez, el programa Java puede emplear el método toPath() para construir el nombre. Utilizar este método es más eficaz que crear un objeto QSYSObjectPathName.

```
  // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree un objeto llamada a programa.
ProgramCall pgm = new ProgramCall(sys);

    // Utilice el método toPath para crear
    // el nombre que representa el programa
    // PRINT_IT de la biblioteca REPORTS.
pgm.setProgram(QSYSObjectPathName.toPath("REPORTS",
                                        "PRINT_IT",
                                        "PGM"));

    // ...Ejecute el programa, procese
    // los resultados.
```

Ejemplo 3: en este ejemplo, se ha proporcionado a un programa Java una vía de acceso del sistema de archivos integrado. Puede utilizarse la clase QSYSObjectPathName para analizar este nombre en sus componentes:

```
    // Cree un objeto nombre de vía de acceso
    // a partir del nombre de sistema de archivos
    // integrado totalmente calificado.
QSYSObjectPathName ifsName = new QSYSObjectPathName(pathName);

    // Utilice el objeto nombre de vía para
    // obtener la biblioteca, el nombre y
    // el tipo del objeto servidor.
String library = ifsName.getLibraryName();
String name    = ifsName.getObjectName();
String type    = ifsName.getObjectType();
```

Acceso a nivel de registro

Las clases de acceso a nivel de registro proporcionan la posibilidad de realizar las tareas siguientes:

- Crear un archivo físico de iSeries especificando uno de estos elementos:
  - La longitud del registro
  - Un archivo fuente DDS (especificaciones de descripción de datos)
  - Un objeto RecordFormat
- Recuperar el formato de registro de un archivo físico o lógico de iSeries o los formatos de registro de un archivo lógico de formato múltiple de iSeries.

  Nota: el formato de registro del archivo no se recupera en su totalidad. Los formatos de registro recuperados están destinados a utilizarse cuando se establece el formato de registro para un objeto AS400File. Solo se recupera la información suficiente para describir el contenido de un registro del archivo. No se recupera información de formato de registro tal como las cabeceras de columna y los alias.
- Acceder a los registros de un archivo de iSeries secuencialmente, por número de registro o por clave.
- Escribir registros en un archivo de iSeries.
- Actualizar registros en un archivo de iSeries secuencialmente, por número de registro o por clave.
- Suprimir registros de un archivo de iSeries secuencialmente, por número de registro o por clave.
- Bloquear un archivo de iSeries para distintos tipos de acceso.
- Utilizar el control de compromiso para permitir a un programa Java efectuar estas operaciones:
  - Iniciar el control de compromiso para la conexión.
  - Especificar distintos niveles de bloqueo de control de compromiso para los distintos archivos.
  - Comprometer y retrotraer transacciones.
- Suprimir archivos de iSeries.
- Suprimir un miembro de un archivo de iSeries.

Nota: las clases de acceso a nivel de registro no dan soporte a archivos lógicos de unión ni a campos clave nulos.

A continuación figuran algunas clases y las funciones que llevan a cabo:

- AS400File es la clase base abstracta de las clases de acceso a nivel de registro. Proporciona métodos que permiten acceder secuencialmente a los registros, crear y suprimir archivos y miembros, y realizar las actividades de control de compromiso.
- La clase KeyedFile representa un archivo de iSeries al que se accede por clave.
- La clase SequentialFile representa un archivo de iSeries al que se accede por número de registro.
- La clase AS400FileRecordDescription proporciona los métodos que permiten recuperar el formato de registro de un archivo de iSeries.

Las clases de acceso a nivel de registro requieren un objeto AS400 que representa el sistema que tiene los archivos de base de datos. Al utilizar las clases de acceso a nivel de registro, el objeto AS400 se conecta al iSeries. En Gestión de conexiones encontrará información sobre cómo se gestionan las conexiones.

Las clases de acceso a nivel de registro requieren el nombre de vía de acceso del sistema de archivos integrado del archivo de base de datos. En Nombres de vía de acceso del sistema de archivos integrado encontrará más información.

Las clases de acceso a nivel de registro utilizan:

- La clase RecordFormat para describir un registro del archivo de base de datos
- La clase Record para proporcionar acceso a los registros del archivo de base de datos

- La clase LineDataRecordWriter para escribir un registro en formato de datos de línea

Estas clases se describen en la sección Conversión de datos.

Ejemplos

- El ejemplo de acceso secuencial muestra cómo se accede secuencialmente a un archivo de iSeries.
- El ejemplo de leer un archivo muestra cómo se utilizan las clases de acceso a nivel de registro para leer un archivo de iSeries.
- El ejemplo de archivo por clave muestra cómo se utilizan las clases de acceso a nivel de registro para leer registros por clave en un archivo de iSeries.

AS400File

La clase AS400File proporciona los métodos que permiten realizar las tareas siguientes:

- Crear y suprimir archivos y miembros físicos del servidor
- Leer y escribir registros en archivos del servidor
- Bloquear archivos para distintos tipos de acceso
- Utilizar bloques de registros para aumentar el rendimiento
- Fijar la posición del cursor dentro de un archivo abierto del servidor
- Gestionar las actividades de control de compromiso

KeyedFile

La clase KeyedFile proporciona a un programa Java el acceso por clave a un archivo del servidor. Acceso por clave quiere decir que el programa Java tan solo ha de especificar una clave para acceder a los registros de un archivo. Hay métodos para situar el cursor, leer, actualizar y suprimir registros por clave.

Para situar el cursor, utilice estos métodos:

- positionCursor(Object[]) - sitúa el cursor en el primer registro que tiene la clave especificada.
- positionCursorAfter(Object[]) - sitúa el cursor en el registro que hay después del primer registro que tiene la clave especificada.
- positionCursorBefore(Object[]) - sitúa el cursor en el registro que hay antes del primer registro que tiene la clave especificada.

Para suprimir un registro, utilice este método:

- deleteRecord(Object[]) - suprime el primer registro que tiene la clave especificada.

Los métodos de lectura son:

- read(Object[]) - lee el primer registro que tiene la clave especificada.
- readAfter(Object[]) - lee el registro situado después del primer registro que tiene la clave especificada.
- readBefore(Object[]) - lee el registro situado antes del primer registro que tiene la clave especificada.
- readNextEqual() - lee el siguiente registro cuya clave coincide con la clave especificada. La búsqueda empieza a partir del registro situado después de la posición actual del cursor.
- readPreviousEqual() - lee el registro anterior cuya clave coincide con la especificada.La búsqueda empieza a partir del registro situado antes de la posición actual del cursor.

Para actualizar un registro, utilice este método:

- update(Object[]) - actualiza el registro que tiene la clave especificada.

También se proporcionan métodos para especificar criterios de búsqueda cuando se sitúa, lee y actualiza por clave. Los valores válidos de los criterios de búsqueda son:

- Igual - se busca el primer registro cuya clave coincide con la especificada.
- Menor que - se busca el último registro cuya clave se encuentra antes de la especificada según el orden de las claves del archivo.
- Igual o menor que - se busca el primer registro cuya clave coincide con la especificada.Si no se encuentra ninguno, se busca el último registro cuya clave se halla antes de la especificada según el orden de las claves del archivo.
- Mayor que - se busca el primer registro cuya clave se encuentra después de la especificada según el orden de las claves del archivo.
- Igual o mayor que - se busca el primer registro cuya clave coincide con la especificada.Si no se encuentra ninguno, se busca el primer registro cuya clave se halla después de la especificada según el orden de las claves del archivo.

KeyedFile es una subclase de AS400File; todos los métodos de AS400File están disponibles en KeyedFile.

Especificar la clave

La clave de un objeto KeyedFile se representa mediante una matriz de objetos Java cuyos tipos y orden se corresponden con los tipos y el orden de los campos clave tal como los especifica el objeto RecordFormat para el archivo.

El ejemplo que sigue muestra cómo se especifica la clave para el objeto KeyedFile.

```
        // Especifique la clave para un archivo cuyos campos clave,
        // puestos por orden, son:
        //    CUSTNAME    CHAR(10)
        //    CUSTNUM     BINARY(9)
        //    CUSTADDR    CHAR(100)VARLEN()
        // Observe que el último es un campo de longitud variable.
    Object[] theKey = new Object[3];
    theKey[0] = "John Doe";
    theKey[1] = new Integer(445123);
    theKey[2] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

Un objeto KeyedFile acepta claves parciales, así como claves completas. No obstante, los valores de los campos clave se han de especificar por orden.

Por ejemplo:

```
        // Especifique una clave parcial para un archivo cuyos campos
clave,
        // puestos por orden, son:
        //    CUSTNAME    CHAR(10)
        //    CUSTNUM     BINARY(9)
        //    CUSTADDR    CHAR(100)VARLEN()
    Object[] partialKey = new Object[2];
    partialKey[0] = "John Doe";
    partialKey[1] = new Integer(445123);

        // Ejemplo de una clave parcial NO VÁLIDA.
    Object[] INVALIDPartialKey = new Object[2];
    INVALIDPartialKey[0] = new Integer(445123);
    INVALIDPartialKey[1] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

No se da soporte a las claves nulas ni a los campos clave nulos.

Los valores de campo clave para un registro se pueden obtener del objeto Record correspondiente a un archivo mediante el método getKeyFields().

El ejemplo siguiente muestra cómo se lee en un archivo por clave:

```
        // Cree un objeto AS400; el archivo
        // existe en este servidor.
    AS400 sys = new AS400("mySystem.myCompany.com");

        // Cree un objeto archivo que represente el archivo.
    KeyedFile myFile = new KeyedFile(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

        // Supongamos que se ha empleado la clase
        // AS400FileRecordDescription para generar el código para una
        // subclase de RecordFormat que representa el formato de registro
        // del archivo MYFILE de la biblioteca MYLIB. El código se ha
        // compilado y está disponible para que lo utilice un programa
Java.
    RecordFormat recordFormat = new MYKEYEDFILEFormat();

        // Establezca el formato de registro para myFile.
```

```
        // Esto se debe realizar antes de invocar el método open().
     myFile.setRecordFormat(recordFormat);

        // Abra el archivo.
     myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

        // El formato de registro del archivo contiene
        // cuatro campos clave, CUSTNUM, CUSTNAME, PARTNUM
        // y ORDNUM, en este orden.
        // partialKey contendrá 2 valores de campo clave.
        // Debido a que los valores de campo clave deben estar
        // ordenados, partialKey constará de valores para
        // CUSTNUM y CUSTNAME.
     Object[] partialKey = new Object[2];
     partialKey[0] = new Integer(1);
     partialKey[1] = "John Doe";

        // Lea el primer registro que coincida con partialKey.
     Record keyedRecord = myFile.read(partialKey);

        // De no encontrarse el registro, se devuelve null.
     if (keyedRecord != null)
     { // Se ha encontrado el registro correspondiente a John Doe; imprima
la información.
       System.out.println("Información de cliente " + (String)partialKey[1]
+ ":");
       System.out.println(keyedRecord);
     }

                    ....

        // Cierre el archivo porque ya ha terminado de usarlo.
     myFile.close();

        // Desconecte, puesto que ya ha terminado de usar el acceso a nivel
de registro.
     sys.disconnectService(AS400.RECORDACCESS);
```

SequentialFile

La clase SequentialFile permite a un programa Java acceder a un archivo del servidor por número de registro. Hay métodos para situar el cursor, leer, actualizar y suprimir registros por número de registro.

Para situar el cursor, utilice estos métodos:

- positionCursor(int) - sitúa el cursor en el registro que tiene el número de registro especificado.
- positionCursorAfter(int) - sitúa el cursor en el registro posterior al que tiene el número de registro especificado.
- positionCursorBefore(int) - sitúa el cursor en el registro anterior al que tiene el número de registro especificado.

Para suprimir un registro, utilice este método:

- deleteRecord(int) - suprime el registro que tiene el número de registro especificado.

Para leer un registro, utilice estos métodos:

- read(int) - lee el registro que tiene el número de registro especificado.
- readAfter(int) - lee el registro posterior al que tiene el número de registro especificado.
- readBefore(int) - lee el registro anterior al que tiene el número de registro especificado.

Para actualizar un registro, utilice este método:

- update(int) - actualiza el registro que tiene el número de registro especificado.

SequentialFile es una subclase de AS400File; todos los métodos de AS400File están disponibles en SequentialFile.

El ejemplo siguiente muestra cómo se utiliza la clase SequentialFile:

```
        // Cree un objeto AS400; el archivo
        // existe en este servidor.
    AS400 sys = new AS400("mySystem.myCompany.com");

        // Cree un objeto archivo que represente el archivo.
    SequentialFile myFile = new SequentialFile(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

        // Supongamos que se ha empleado la clase
        // AS400FileRecordDescription para generar el código para una
        // subclase de RecordFormat que representa el formato de registro
        // del archivo MYFILE de la biblioteca MYLIB. El código se ha
        // compilado y está disponible para que lo utilice un programa
Java.
    RecordFormat recordFormat = new MYFILEFormat();

        // Establezca el formato de registro para myFile.
        // Esto se debe realizar antes de invocar el método open().
    myFile.setRecordFormat(recordFormat);

        // Abra el archivo.
    myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

        // Suprima el registro número 2.
    myFile.delete(2);

        // Lea el registro número 5 y actualícelo.
```

```
     Record updateRec = myFile.read(5);
     updateRec.setField("CUSTNAME", newName);

         // Utilice el método update() de la clase base,
         // puesto que ya está situado en el registro.
     myFile.update(updateRec);

         // Actualice el registro número 7.
     updateRec.setField("CUSTNAME", nextNewName);
     updateRec.setField("CUSTNUM", new Integer(7));
     myFile.update(7, updateRec);

                    ....

         // Cierre el archivo porque ya ha terminado de usarlo.
     myFile.close();

         // Desconecte, puesto que ya ha terminado de usar el acceso a nivel
de registro.
     sys.disconnectService(AS400.RECORDACCESS);
```

AS400FileRecordDescription

La clase AS400FileRecordDescription proporciona los métodos que permiten recuperar el formato de registro de un archivo del servidor. Esta clase proporciona métodos para crear código fuente Java para las subclases de RecordFormat y devolver objetos RecordFormat, que describen los formatos de registro de los archivos físicos o lógicos del servidor especificados por el usuario. La salida de estos métodos puede utilizarse como entrada para un objeto AS400File al establecer el formato de registro.

Se recomienda utilizar siempre la clase AS400FileRecordDescription para generar el objeto RecordFormat cuando el archivo ya existe en el servidor.

Nota: la clase AS400FileRecordDescription no recupera la totalidad del formato de registro de un archivo. Solo se recupera la información suficiente para describir el contenido de los registros que componen el archivo. No se recupera información como la de las cabeceras de columna, los alias y los campos de referencia. Por lo tanto, los formatos de registro recuperados no necesariamente crean un archivo cuyo formato de registro sea idéntico al del archivo del que se recuperó.

Crear código fuente Java para subclases de RecordFormat para representar el formato de registro de archivos del servidor

El método createRecordFormatSource() crea archivos fuente Java para subclases de la clase RecordFormat. Los archivos se pueden compilar y, luego, una aplicación o un applet los puede utilizar como entrada para el método AS400File.setRecordFormat().

El método createRecordFormatSource() debe utilizarse como herramienta en tiempo de desarrollo para recuperar los formatos de registro de archivos existentes en el servidor. Con este método, el fuente correspondiente a la subclase de la clase RecordFormat puede crearse una vez, modificarse si es necesario, compilarse y después lo pueden utilizar muchos programas Java que accedan a los mismos archivos del servidor. Únicamente las aplicaciones Java pueden utilizar este método, porque crea los archivos en el sistema local. Sin embargo, la salida (el código fuente Java) se puede compilar y después la pueden utilizar tanto las aplicaciones como los applets Java.

Nota: este método sobrescribe los archivos cuyo nombre sea idéntico al de los archivos fuente Java que se crean.

Ejemplo 1: el ejemplo siguiente muestra cómo se utiliza el método createRecordFormatSource():

```
         // Cree un objeto AS400; el archivo
         // existe en este servidor.
     AS400 sys = new AS400("mySystem.myCompany.com");

         // Cree un objeto AS400FileRecordDescription que represente el
archivo.
     AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

         // Cree el archivo fuente Java en el directorio de trabajo actual.
         // Especifique "package com.myCompany.myProduct;" para la
         // sentencia package en el fuente, ya que la clase se va a
suministrar
         // como parte de "my product".
     myFile.createRecordFormatSource(null, "com.myCompany.myProduct");

         // Suponiendo que el nombre de formato del archivo MYFILE sea
FILE1, el
         // archivo FILE1Format.java se creará en el directorio de trabajo
actual.
         // Va a sobrescribir cualquier archivo que tenga el mismo nombre.
El
         // nombre de la clase será FILE1Format. La clase se derivará de
RecordFormat.
```

Ejemplo 2: compile el archivo creado más arriba, FILE1Format.java, y utilícelo de la siguiente manera:

```
        // Cree un objeto AS400; el archivo
        // existe en este servidor.
    AS400 sys = new AS400("mySystem.myCompany.com");

        // Cree un objeto AS400File que represente el archivo.
    SequentialFile myFile = new SequentialFile(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

        // Establezca el formato de registro.
        // Ello presupone que import.com.myCompany.myProduct.FILE1Format;
        // ya se ha realizado.

    myFile.setRecordFormat(new FILE1Format());

        // Abra el archivo y léalo.
                    ....

        // Cierre el archivo porque ya ha terminado de usarlo.
    myFile.close();

        // Desconecte, puesto que ya ha terminado de usar el acceso a nivel
de registro.
    sys.disconnectService(AS400.RECORDACCESS);
```

Crear objetos RecordFormat para representar el formato de registro de archivos del servidor

El método retrieveRecordFormat() devuelve una matriz de objetos RecordFormat que representan los formatos de registro de un archivo existente en el servidor. Lo más habitual es que en la matriz solo se devuelva un objeto RecordFormat. Cuando el archivo cuyo formato de registro se está recuperando es un archivo lógico de formato múltiple, se devuelve más de un objeto RecordFormat. Utilice este método para recuperar dinámicamente en tiempo de ejecución el formato de registro de un archivo existente en el servidor. El objeto RecordFormat puede usarse luego como entrada para el método AS400File.setRecordFormat().

El ejemplo siguiente muestra cómo se utiliza el método retrieveRecordFormat():

```
        // Cree un objeto AS400; el archivo
        // existe en este servidor.
    AS400 sys = new AS400("mySystem.myCompany.com");

        // Cree un objeto AS400FileRecordDescription que represente el
archivo.
    AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

        // Recupere el formato de registro correspondiente al archivo.
    RecordFormat[] format = myFile.retrieveRecordFormat();

        // Cree un objeto AS400File que represente el archivo.
    SequentialFile myFile = new SequentialFile(sys,
"/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

        // Establezca el formato de registro.
    myFile.setRecordFormat(format[0]);
```

```
         // Abra el archivo y léalo.
                        ....

         // Cierre el archivo porque ya ha terminado de usarlo.
     myFile.close();

         // Desconecte, puesto que ya ha terminado de usar el acceso a nivel
de registro.
     sys.disconnectService(AS400.RECORDACCESS);
```

Llamada a programa de servicio

La clase ServiceProgramCall le permite llamar a un programa de servicio de iSeries. ServiceProgramCall es una subclase de la clase ProgramCall que se utiliza para llamar a programas de iSeries. Si desea llamar a un programa de iSeries, hágalo con la clase ProgramCall.

La clase ServiceProgramCall hace que sea posible llamar a un programa de servicio de iSeries, pasar datos a un programa de servicio de iSeries mediante parámetros de entrada y acceder a los datos devueltos por el programa de servicio de iSeries mediante parámetros de salida. La utilización de ServiceProgramCall hace que el objeto AS400 se conecte al iSeries. En Gestión de conexiones encontrará información acerca de cómo se gestionan las conexiones.

El comportamiento por omisión consiste en el caso de los programas de servicio en que la ejecución se lleve a cabo en un trabajo servidor aparte, aun cuando el programa Java y el programa de servicio estén en el mismo servidor. Puede alterar temporalmente el comportamiento por omisión y hacer que el programa de servicio se ejecute en el trabajo Java mediante el método setThreadSafe() heredado (de ProgramCall).

Utilización de la clase ServiceProgramCall

Para poder utilizar la clase ServiceProgramCall, debe asegurarse de que se cumplen estos requisitos:

- El programa de servicio debe estar en un servidor AS/400e o iSeries que ejecute OS/400 V4R4 o posterior
- No puede pasar más de siete parámetros al programa de servicio
- El valor de retorno del programa de servicio es vacío (void) o de tipo numérico

Trabajar con objetos ProgramParameter

La clase ProgramParameter funciona junto con la clase ServiceProgramCall para pasar datos de parámetro a un programa de servicio de iSeries o desde él. Para pasar datos de entrada al programa de servicio de iSeries, utilice setInputData().

Para solicitar la cantidad de datos de salida que desea que se devuelva, utilice setOutputDataLength(). Para recuperar los datos de salida una vez que el programa de servicio ha terminado de ejecutarse, utilice getOutputData(). La clase ServiceProgramCall, además de conocer los datos en sí, necesita conocer cómo ha de pasar los datos de parámetro al programa de servicio. El método setParameterType() de ProgramParameter permite proporcionar esta información. El tipo indica si el parámetro se pasa por valor o por referencia. En los dos casos, los datos se envían desde el cliente al servidor. Una vez que los datos estén en el iSeries, el servidor utiliza el tipo de parámetro para llamar correctamente al programa de servicio.

Todos los parámetros tendrán el formato de una matriz de bytes. Por lo tanto, para realizar la conversión entre los formatos de iSeries y Java, se utilizan las clases de conversión y descripción de datos.

Clases SystemStatus

Las clases SystemStatus permiten recuperar información sobre el estado del sistema, así como recuperar y cambiar información de agrupación del sistema. El objeto SystemStatus dispone de los siguientes métodos para recuperar información sobre el estado del sistema:

- getUsersCurrentSignedOn(): devuelve el número de usuarios que tienen iniciada una sesión en el sistema en este momento
- getUsersTemporarilySignedOff(): devuelve el número de trabajos interactivos desconectados
- getDateAndTimeStatusGathered(): devuelve la fecha y la hora del momento en que se recopiló información sobre el estado del sistema
- getJobsInSystem(): devuelve el número total de trabajos de usuario y del sistema que se están ejecutando en este momento
- getBatchJobsRunning(): devuelve el número de trabajos por lotes que se ejecutan actualmente en el sistema
- getBatchJobsEnding(): devuelve el número de trabajos por lotes que están en proceso de finalización
- getSystemPools(): devuelve una enumeración que contiene un objeto SystemPool para cada una de las agrupaciones del sistema

Además de los métodos que hay en la clase SystemStatus, también puede acceder a SystemPool mediante SystemStatus. SystemPool le permite obtener información acerca de las agrupaciones del sistema y realizar cambios en dicha información.

Ejemplo

Este ejemplo muestra cómo se utiliza la puesta en antememoria con la clase SystemStatus:

```
AS400 system = new AS400("MyAS400");
SystemStatus status = new SystemStatus(system);

// Active la puesta en antememoria. Por omisión, está desactivada.
status.setCaching(true);

// Esto recuperará el valor del sistema.
// Cada llamada ulterior utilizará el valor puesto en antememoria,
// en vez de recuperarlo del sistema.
int jobs = status.getJobsInSystem();


// ...Realice aquí otras operaciones...


// Así se averigua si la puesta en antememoria todavía está habilitada.
if (status.isCaching())
{
// Esto recuperará el valor de la antememoria.
  jobs = status.getJobsInSystem();
}

// Vaya al sistema la próxima vez, aunque la puesta en antememoria esté
habilitada.
status.refreshCache();

// Esto recuperará el valor del sistema.
jobs = status.getJobsInSystem();
```

```
// Desactive la puesta en antememoria. Todas las llamadas ulteriores irán al
sistema.
status.setCaching(false);

// Esto recuperará el valor del sistema.
jobs = status.getJobsInSystem();
```

Valores del sistema

Las clases de valores del sistema permiten a un programa Java recuperar y cambiar valores del sistema y atributos de red. También puede definir su propio grupo para contener los valores del sistema que desee.

Un objeto SystemValue principalmente contiene la información siguiente:

- Nombre
- Descripción
- Release
- Valor

Mediante la clase SystemValue, recupere un único valor del sistema con el método getValue() y cambie un valor del sistema con el método setValue().

Asimismo, puede recuperar información de grupo sobre un valor del sistema determinado:

- Para recuperar el grupo definido por el sistema al que pertenece un valor del sistema, utilice el método getGroup().
- Para recuperar el grupo definido por el usuario al que pertenece un objeto SystemValue (si existe), utilice los métodos getGroupName() y getGroupDescription().

El valor de un valor del sistema, siempre que se recupera por primera vez, se obtiene a partir del iSeries y se pone en antememoria. En las recuperaciones ulteriores, el valor que se devuelve es el que está en la antememoria. Si en vez del valor almacenado en la antememoria, se desea obtener el valor actual del iSeries, se debe utilizar un método clear() para borrar la antememoria actual.

Lista de valores del sistema

SystemValueList representa una lista de valores del sistema del sistema iSeries especificado. La lista se subdivide en varios grupos definidos por el sistema que permiten al programa Java acceder simultáneamente a parte de los valores del sistema.

Grupo de valores del sistema

SystemValueGroup representa un conjunto definido por el usuario de valores del sistema y atributos de red. No se trata de un contenedor sino de una fábrica para generar y mantener conjuntos exclusivos de valores del sistema.

Puede crear un objeto SystemValueGroup especificando uno de los grupos definidos por el sistema (una de las constantes de la clase SystemValueList) o especificando una matriz de nombres de valores del sistema.

Puede añadir de forma individual los nombres de valores del sistema para incluir en el grupo mediante el método add(). Asimismo, puede eliminarlos mediante el método remove().

Una vez que el objeto SystemValueGroup se haya llenado con los nombres de valores del sistema deseados, obtenga los objetos SystemValue reales del grupo llamando al método getSystemValues(). De esta forma, un objeto SystemValueGroup toma un conjunto de nombres de valores del sistema y genera un vector de objetos SystemValue, todos ellos con el sistema, el nombre de grupo y la descripción de grupo del objeto SystemValueGroup.

Para renovar un vector de todos los objetos SystemValue a la vez, utilice el método refresh().

Ejemplos de cómo se utilizan las clases SystemValue y SystemValueList

El ejemplo siguiente muestra cómo se crea y recupera un valor del sistema:

```
//Cree un objeto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

//Cree un objeto valor del sistema que represente los segundos actuales en
el sistema.
SystemValue sysval = new SystemValue(sys, "QSECOND");


//Recupere el valor.
String second = (String)sysval.getValue();


//En este momento, QSECOND se pone en antememoria. Borre la antememoria para
recuperar el
//valor más actualizado del sistema.
sysval.clear();
second = (String)sysval.getValue();

//Cree una lista de valores del sistema.
SystemValueList list = new SystemValueList(sys);


//Recupere la totalidad de los valores de fecha/hora del sistema.
Vector vec = list.getGroup(SystemValueList.GROUP_DATTIM);


//Desconéctese del sistema.
sys.disconnectAllServices();
```

Ejemplos de cómo se utiliza la clase SystemValueGroup

El ejemplo siguiente muestra cómo se construye un grupo de nombres de valores del sistema y cómo después se trabaja con ellos:

```
//Cree un objeto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

//Cree un grupo de valores del sistema que inicialmente representa todos los
atributos de red del sistema.
String name = "My Group";
String description = "Es uno de mis valores del sistema.";
SystemValueGroup svGroup = new SystemValueGroup(sys, name, description,
SystemValueList.GROUP_NET);

//Añada al grupo algunos nombres de valores del sistema más y elimine
algunos no deseados.
svGroup.add("QDATE");
svGroup.add("QTIME");
svGroup.remove("NETSERVER");
svGroup.remove("SYSNAME");

//Obtenga los objetos SystemValue reales. Se devuelven dentro de un vector.
Vector sysvals = svGroup.getSystemValues();
```

```
//Observará que es uno de los valores indicados como mis valores del
sistema.
SystemValue mySystemValue = (SystemValue)sysvals.elementAt(0);
System.out.println(mySystemValue.getName()+" -
"+mySystemValue.getGroupDescription());

//Podemos añadir al grupo otro objeto SystemValue de otro sistema.
AS400 sys2 = new AS400("otherSystem.myCompany.com");
SystemValue sv = new SystemValue(sys2, "QDATE");
sysvals.addElement(sv);

//Ahora renueve el grupo completo de todos los valores del sistema a la vez.
//No importa si algunos valores del sistema son de sistemas iSeries
distintos.
//No importa si algunos valores del sistema se han generado con
SystemValueGroup y otros no.
SystemValueGroup.refresh(sysvals);

//Desconéctese de los sistemas.
sys.disconnectAllServices();
sys2.disconnectAllServices();
```

Trace

El objeto Trace permite al programa Java anotar puntos de rastreo y mensajes de diagnóstico. Esta información ayuda a reproducir y a diagnosticar problemas.

Nota: también puede establecer el rastreo mediante las propiedades de rastreo del sistema.

La clase Trace anota las siguientes categorías de información:

Categoría de información	Descripción
Conversión	Anota las conversiones de juego de caracteres entre las páginas de códigos y Unicode. Sólo las clases de IBM Toolbox para Java deben utilizar esta categoría.
Corriente de datos	Anota los datos que fluyen entre el iSeries y el programa Java. Sólo las clases de IBM Toolbox para Java deben utilizar esta categoría.
Diagnóstico	Anota información sobre el estado.
Error	Anota errores adicionales que ocasionan una excepción.
Información	Rastrea el flujo a través de un programa.
»PCML	Esta categoría se utiliza para determinar cómo interpreta PCML los datos que se envían al servidor y que se reciben del mismo.«
Proxy	Las clases de IBM Toolbox para Java utilizan esta categoría para anotar el flujo de datos entre el cliente y el servidor proxy.
Aviso	Anota información acerca de los errores de los que el programa ha podido recuperarse.
Total	Esta categoría permite habilitar o inhabilitar el rastreo para todas las categorías anteriores a la vez. La información de rastreo no se puede anotar directamente en esta categoría.

Las clases de IBM Toolbox para Java también utilizan las categorías de rastreo. Cuando un programa Java habilita las anotaciones, la información de IBM Toolbox para Java se incluye junto con la información registrada por la aplicación.

El rastreo se puede habilitar para una sola categoría o para un conjunto de categorías. Una vez seleccionadas las categorías, utilice el método setTraceOn para activar y desactivar el rastreo. Para escribir los datos en las anotaciones, se utiliza el método log.

Puede enviar los datos de rastreo de distintos componentes a anotaciones separadas. Los datos de rastreo, por omisión, se escriben en las anotaciones por omisión. Utilice el rastreo de componentes para escribir datos de rastreo específicos de la aplicación en unas anotaciones distintas o en la salida estándar. El rastreo de componentes permite separar fácilmente los datos de rastreo de una aplicación específica de los demás datos.

Una cantidad excesiva de anotaciones puede afectar al rendimiento. Utilice el método isTraceOn para consultar el estado actual del rastreo. El programa Java puede emplear este método para determinar si debe construir el registro de rastreo antes de llamar al método log. Llamar al método log cuando el rastreo está desactivado no es un error, pero se invierte más tiempo.

El valor por omisión es escribir información de anotaciones en la salida estándar. Para redirigir las anotaciones a un archivo, llame al método setFileName() desde la aplicación Java. En general, esto solo funciona para las aplicaciones Java porque la mayoría de los navegadores no dan acceso a los applets para escribir en el sistema de archivos local.

Las anotaciones están desactivadas por omisión. Los programas Java deben proporcionar al usuario un procedimiento que le permita activar las anotaciones para que le resulte fácil habilitarlas. Por ejemplo, la aplicación puede realizar un análisis para obtener un parámetro de línea de mandatos que indique qué categoría de datos debe anotarse. El usuario puede establecer este parámetro cuando se necesite información de anotaciones.

Los ejemplos que hay a continuación muestran cómo se utiliza la clase Trace.

Ejemplo 1: el siguiente es un ejemplo de cómo se utiliza el método setTraceOn y de cómo se escriben datos en un archivo de anotaciones mediante el método log.

```
        // Habilite las anotaciones de diagnóstico, información y aviso.
     Trace.setTraceDiagnosticOn(true);
     Trace.setTraceInformationOn(true);
     Trace.setTraceWarningOn(true);

     // Active el rastreo.
     Trace.setTraceOn(true);

     // ...En este punto del programa Java, escriba en las anotaciones.
     Trace.log(Trace.INFORMATION, "Acaba de entrarse en la clase xxx, método
xxx");

        // Desactive el rastreo.
     Trace.setTraceOn(false);
```

Ejemplo 2: los ejemplos que hay a continuación muestran cómo se utiliza el rastreo. El método 2 es la manera preferible de escribir código que utilice el rastreo.

```
     // Método 1 - construya un registro de rastreo
     // y luego llame al método log y deje que la clase de rastreo determine
     // si los datos deben anotarse. Este método funcionará pero será más
lento que
     // el código siguiente.
     String traceData = new String("Acaba de entrarse en la clase xxx, datos
= ");
        traceData = traceData + data + "estado = " + state;
        Trace.log(Trace.INFORMATION, traceData);

     // Método 2 - compruebe el estado de las anotaciones antes de
incorporar la
     // información a las anotaciones. Este método es más rápido cuando el
rastreo no está activo.
     if (Trace.isTraceOn() && Trace.isTraceInformationOn())
     {
     String traceData = new String("acaba de entrarse en la clase xxx, datos
= ");
        traceData = traceData + data + "estado = " + state;
        Trace.log(Trace.INFORMATION, traceData);
     }
```

Ejemplo 3: el ejemplo siguiente muestra cómo se puede utilizar el rastreo de componentes.

```
     // Cree una serie de componente. Es más eficaz crear un
     // objeto que muchos literales String.
     String myComponent1 = "com.myCompany.xyzComponent";
     String myComponent2 = "com.myCompany.abcComponent";

     // Envíe los datos de rastreo de la Caja de Herramientas y los de
componente a archivos distintos.
     // El rastreo de la Caja de Herramientas contendrá toda la información
de rastreo, mientras
     // que cada uno de los archivos de anotaciones de componente sólo
contendrá la información de rastreo
     // específica de ese componente. Si no se especifica un archivo de
rastreo, todos los datos de rastreo
```

```
     // irán a la salida estándar con el componente especificado frente a
cada uno
     // de los mensajes de rastreo.

     // Trace.setFileName("c:\\bit.bucket");
     // Trace.setFileName(myComponent1, "c:\\Component1.log");
     // Trace.setFileName(myComponent2, "c:\\Component2.log");

     Trace.setTraceOn(true);             // Active el rastreo.
     Trace.setTraceInformationOn(true);  // Habilite los mensajes
informativos.

     // Anote los datos de rastreo específicos de componente o los datos de
     // rastreo generales de la Caja de Herramientas.

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

El resultado del ejemplo, si no especifica ningún archivo de rastreo, tendrá este aspecto:

```
     Toolbox for Java - Version 5 Release 1 Modification level 0
     [com.myCompany.xyzComponent] Tue Oct 24 16:02:44 CDT 2000 I am here
     [com.myCompany.abcComponent] Tue Oct 24 16:02:44 CDT 2000 I am there
     Tue Oct 24 16:02:44 CDT 2000 I am everywhere
```

Usuarios y grupos

Las clases de usuarios y grupos permiten obtener una lista de los usuarios y grupos de usuarios existentes en el sistema iSeries, así como información acerca de cada usuario mediante un programa Java.

Nota: Toolbox para Java también proporciona clases de recursos que presentan una infraestructura genérica y una interfaz de programación coherente para trabajar con una gran variedad de objetos y listas de iSeries. Tras leer la información acerca de las clases del paquete access y el paquete de recursos, puede elegir el objeto más adecuado para su aplicación. Las clases de recursos para trabajar con usuarios son RUser y RUserList.

La información de usuario que se puede recuperar comprende la fecha del inicio de sesión anterior, el estado, la fecha del último cambio de contraseña, la fecha de caducidad de la contraseña y la clase de usuario. Cuando acceda al objeto User, debe utilizar el método setSystem() para establecer el nombre del sistema y el método setName() para establecer el nombre de usuario. Tras estos pasos, utilice el método loadUserInformation() para obtener la información del iSeries.

El objeto UserGroup representa un usuario especial cuyo perfil de usuario es un perfil de grupo. Con el método getMembers() puede obtenerse una lista de los usuarios que son miembros del grupo.

El programa Java puede iterar por la lista utilizando una enumeración. Todos los elementos de la enumeración son objetos User; por ejemplo:

```
   // Cree un objeto AS400.
AS400 system = new AS400 ("mySystem.myCompany.com");

// Cree el objeto UserList.
UserList userList = new UserList (system);

// Obtenga la lista de todos los usuarios y grupos.
Enumeration enum = userList.getUsers ();

// Itere por la lista.
  while (enum.hasMoreElements())
{
    User u = (User) enum.nextElement ();
    System.out.println  (u);
}
```

Recuperar información acerca de los usuarios y grupos

Utilice un objeto UserList para obtener una lista de:

- Todos los usuarios y grupos
- Sólo grupos
- Todos los usuarios que son miembros de algún grupo
- Todos los usuarios que no son miembros de ningún grupo

La única propiedad que debe establecerse del objeto UserList es el objeto AS400 que representa el sistema del que se debe recuperar la lista de usuarios.

Por omisión, se devuelven todos los usuarios. Utilice una combinación de los métodos setUserInfo() y setGroupInfo() para especificar exactamente qué usuarios deben devolverse.

Ejemplo: cómo se utiliza UserList para listar todos los usuarios de un grupo determinado

Clase UserSpace

La clase UserSpace representa un espacio de usuario en el servidor. Los parámetros obligatorios son el nombre del espacio de usuario y el objeto AS400 que representa el servidor al que pertenece el espacio de usuario. En la clase UserSpace hay métodos que permiten realizar estas tareas:

- Crear un espacio de usuario.
- Suprimir un espacio de usuario.
- Leer en un espacio de usuario.
- Escribir en un espacio de usuario.
- Obtener los atributos de un espacio de usuario. Un programa Java puede obtener los atributos de valor inicial, valor de longitud y ampliación automática de un espacio de usuario.
- Establecer los atributos de un espacio de usuario. Un programa Java puede establecer los atributos de valor inicial, valor de longitud y ampliación automática de un espacio de usuario.

El objeto UserSpace requiere el nombre de vía de acceso del sistema de archivos integrado del programa. En Nombres de vía de acceso del sistema de archivos integrado puede encontrar más información.

Al utilizar la clase UserSpace, el objeto AS400 se conecta al servidor. En Gestión de conexiones hallará información sobre cómo se gestionan las conexiones.

El siguiente ejemplo crea un espacio de usuario y luego escribe datos en él.

```
       // Cree un objeto AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

         // Cree un objeto espacio de usuario.
     UserSpace US = new UserSpace(sys,

"/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");

         // Use el método create para crear el espacio de usuario en
         // el servidor.
     US.create(10240,                          // El tamaño inicial es de
10 K.
               true,                           // Sustituir si el espacio
de usuario ya existe.
               " ",                            // Ningún atributo ampliado.
               (byte) 0x00,                    // El valor inicial es null.
               "Creado por un programa Java",  // Descripción del espacio
de usuario.
               "*USE");                        // La autorización de uso
público sobre el espacio de usuario es *USE.


         // Use el método write para escribir bytes en el espacio de
usuario.
     US.write("Escribir esta serie en el espacio de usuario.", 0);
```

Clases HTML

Las clases HTML de IBM Toolbox para Java ayudan al usuario a llevar a cabo las acciones siguientes:

- Preparar formularios y tablas para páginas HTML.
- Alinear texto.
- Trabajar una gran variedad de códigos HTML.
- Modificar el idioma y la dirección del texto.
- Crear listas ordenadas y sin ordenar.
- Crear listas de archivos y árboles jerárquicos HTML (y los elementos que contienen).
- Añadir atributos de código que no están definidos en las clases HTML (por ejemplo, los atributos bgcolor y style).

Las clases HTML implementan la interfaz HTMLTagElement. Cada clase genera un código HTML para un tipo de elemento específico. El código se puede recuperar con el método getTag() y luego se puede incorporar a cualquier documento HTML. Los códigos que se generan con las clases HTML son coherentes con la especificación de HTML 3.2.

Las clases HTML pueden funcionar conjuntamente con las clases servlet para obtener datos del servidor iSeries. Sin embargo, también se pueden utilizar solas si se suministran los datos de la tabla o del formulario.

Gracias a las clases HTML, resulta más fácil confeccionar formularios, tablas y demás elementos HTML:

- La clase BidiOrdering permite modificar el idioma y la dirección del texto.
- La clase DirFilter permite determinar si un objeto File es un directorio.
- La clase HTMLAlign permite alinear bloques de salida HTML.
- La clase HTMLFileFilter permite determinar si un objeto File es un archivo.
- Con las clases HTMLForm resulta más fácil confeccionar formularios que con los scripts CGI.
- La clase HTMLHeading permite crear códigos de cabecera para las páginas HTML.
- La clase HTMLHyperlink ayuda a crear enlaces dentro de las páginas HTML.
- »La clase HTMLImage permite crear códigos de imagen para las páginas HTML.«
- Las clases HTMLList ayudan a crear listas para las páginas HTML.
- La clase HTMLMeta permite crear códigos meta para las páginas HTML.
- La clase HTMLParameter especifica parámetros disponibles para HTMLServlet.
- La clase HTMLServlet permite crear un elemento include en el lado del servidor.
- Las clases HTMLTable ayudan a confeccionar tablas para las páginas HTML.
- La clase HTMLText permite acceder a las propiedades de los fonts que hay dentro de las páginas HTML.
- Las clases HTMLTree permiten visualizar un árbol jerárquico HTML formado por elementos HTML.
- La clase URLEncoder codifica los delimitadores que se han de utilizar en un URL de tipo serie.
- La clase URLParser permite analizar una serie de URL para examinar el identificador URI, las propiedades y la referencia.

NOTA: el archivo jt400Servlet.jar incluye tanto las clases HTML como las clases Servlet. Debe actualizar la CLASSPATH para que señale al archivo jt400Servlet.jar si desea utilizar las clases del paquete com.ibm.as400.util.html.

Clase BidiOrdering

La clase BidiOrdering representa un código HTML que modifica el idioma y la dirección del texto. Una serie HTML <BDO> requiere dos atributos, uno para el idioma y otro para la dirección del texto.

La clase BidiOrdering permite llevar a cabo estas acciones:

- Obtener y establecer el atributo de idioma.
- Obtener y establecer la dirección del texto.

Para obtener más información acerca de cómo se utiliza el código HTML <BDO>, consulte el sitio Web de W3C  .

Ejemplo: cómo se utiliza BidiOrdering

En el ejemplo siguiente se crea un objeto BidiOrdering y se establece el idioma y la dirección del mismo:

```
// Cree un objeto BidiOrdering y establezca el idioma y la dirección.
BidiOrdering bdo = new BidiOrdering();
bdo.setDirection(HTMLConstants.RTL);
bdo.setLanguage("AR");


// Cree un texto.
HTMLText text = new HTMLText("Un texto en árabe.");
text.setBold(true);

// Añada el texto al objeto BidiOrdering y obtenga el código HTML.
bdo.addItem(text);
bdo.getTag();
```

La sentencia de imprimir genera el siguiente código HTML:

```
<bdo lang="AR" dir="rtl">
  <b>Un texto en árabe.</b>
</bdo>
```

Cuando se utiliza este código en una página HTML, los navegadores que entienden el código <BDO> visualizan el ejemplo de esta forma:

.txeT cibarA emoS

Clase HTMLAlign

La clase HTMLAlign permite alinear secciones del documento HTML, en lugar de alinear elementos individuales, tales como párrafos o cabeceras.

La clase HTMLAlign representa el código <DIV> y el atributo de alineación asociado al mismo. Puede utilizar la alineación a la derecha, a la izquierda o centrada.

Puede emplear esta clase para llevar a cabo diversas acciones, tales como las siguientes:

- Añadir o eliminar elementos de la lista de códigos que desea alinear
- Obtener y establecer la alineación
- Obtener y establecer la dirección de interpretación del texto
- Obtener y establecer el idioma del elemento de entrada
- Obtener una representación de tipo String del objeto HTMLAlign

Ejemplo: crear objetos HTMLAlign

El ejemplo que sigue crea una lista sin ordenar y a continuación crea un objeto HTMLAlign para alinear toda la lista:

```
    // Cree una lista sin ordenar.
    UnorderedList uList = new UnorderedList();
    uList.setType(HTMLConstants.DISC);
    UnorderedListItem uListItem1 = new UnorderedListItem();
    uListItem1.setItemData(new HTMLText("Lista sin ordenar centrada"));
    uList.addListItem(uListItem1);
    UnorderedListItem uListItem2 = new UnorderedListItem();
    uListItem2.setItemData(new HTMLText("Otro elemento"));
    uList.addListItem(uListItem2);

    // Alinee la lista.
    HTMLAlign align = new HTMLAlign(uList, HTMLConstants.CENTER);
    System.out.println(align);
```

El ejemplo anterior genera el código siguiente:

```
    <div align="center">
    <ul type="disc">
      <li>Lista sin ordenar centrada</li>
      <li>Otro elemento</li>
</ul>
```

Este código, cuando se utiliza en una página HTML, ofrece este aspecto:

- Lista sin ordenar centrada
- Otro elemento

Clases de formularios HTML

La clase HTMLForm representa un formulario HTML. Esta clase le permite:

- Añadir un elemento (por ejemplo, un botón, un hiperenlace o una tabla HTML) a un formulario
- Eliminar un elemento de un formulario
- Establecer otros atributos de formulario; por ejemplo, qué método se ha de usar para enviar el contenido del formulario al servidor, la lista de parámetros ocultos o la dirección URL de la acción

El constructor del objeto HTMLForm toma una dirección de URL. A esta dirección la llamamos URL de acción. Es la ubicación de la aplicación en el servidor que va a procesar la entrada del formulario. El URL de acción se puede especificar en el constructor; también se puede establecer la dirección mediante el método setURL(). Los atributos de formulario se establecen con los diversos métodos set y se recuperan con los diversos métodos get.

Los elementos de código HTML se pueden añadir a un objeto HTMLForm utilizando el método addElement() y se pueden eliminar utilizando el método removeElement(). Utilice las siguientes clases de elementos de códigos HTML en los objetos HTMLForm:

- Clases FormInput: representan los elementos de entrada de un formulario HTML
- Clases LayoutFormPanel: representan un diseño de los elementos de formulario de un formulario HTML
- TextAreaFormElement: representa un elemento de área de texto de un formulario HTML
- LabelFormElement: representa una etiqueta para un elemento de formulario HTML
- SelectFormElement: representa un tipo de entrada de selección para un formulario HTML
- SelectOption: representa una opción para un objeto SelectFormElement de un formulario HTML
- RadioFormInputGroup: representa un grupo de objetos de entrada que son botones de selección; el usuario solo puede seleccionar un botón del grupo

Obviamente, puede añadir otros elementos de códigos a un formulario, tales como los siguientes:

- HTMLText
- HTMLHyperlink
- HTMLTable

Si desea más información sobre cómo se utiliza la clase HTMLForm para crear un formulario, consulte este ejemplo y la salida resultante.

Clases FormInput

La clase FormInput permite llevar a cabo estas acciones:

- Obtener y establecer el nombre de un elemento de entrada
- Obtener y establecer el tamaño de un elemento de entrada
- Obtener y establecer el valor inicial de un elemento de entrada

La clase FormInput se amplía mediante las clases de la lista siguiente. Estas clases permiten crear tipos específicos de elementos de entrada de formulario y permiten obtener y establecer diversos atributos o recuperar el código HTML del elemento de entrada:

- ButtonFormInput: representa un elemento que es un botón de un formulario HTML
- FileFormInput: representa un tipo de entrada de archivo de un formulario HTML
- HiddenFormInput: representa un tipo de entrada oculta de un formulario HTML
- ImageFormInput: representa un tipo de entrada de imagen de un formulario HTML
- ResetFormInput: representa una entrada de botón de restablecer de un formulario HTML
- SubmitFormInput: representa una entrada de botón de someter de un formulario HTML
- TextFormInput: representa una entrada de texto de una sola línea de un formulario HTML en la que se define el número máximo de caracteres de una línea. Para un tipo de entrada de contraseña, se utiliza la clase PasswordFormInput, que amplía TextFormInput y representa un tipo de entrada de contraseña de un formulario HTML
- ToggleFormInput: representa un tipo de entrada de conmutador de un formulario HTML. El usuario puede establecer u obtener la etiqueta de texto y especificar si el conmutador se debe marcar o seleccionar. El tipo de entrada de conmutador puede ser uno de estos dos:
  - RadioFormInput: representa un tipo de entrada de botón de selección de un formulario HTML. Los botones de selección se pueden colocar en grupos con la clase RadioFormInputGroup; esta clase crea un grupo de botones de selección donde el usuario sólo selecciona una de las opciones presentadas.
  - CheckboxFormInput: representa un tipo de entrada de recuadro de selección de un formulario HTML en que el usuario puede seleccionar más de una de las opciones presentadas y en que el recuadro de selección se inicializa como marcado o no marcado.

Clase ButtonFormInput

La clase ButtonFormInput representa un elemento que es un botón de un formulario HTML.

El ejemplo siguiente muestra cómo se crea un objeto ButtonFormInput:

```
    ButtonFormInput button = new ButtonFormInput("button1", "Pulse aquí",
"test()");
    System.out.println(button.getTag());
```

Este ejemplo genera el siguiente código:

```
    <input type="button" name="button1" value="Pulse aquí" onclick="test()"
/>
```

Clase FileFormInput

La clase FileFormInput representa un tipo de entrada de archivo de un formulario HTML.

El ejemplo de código siguiente muestra cómo se crea un objeto FileFormInput nuevo:

```
FileFormInput file = new FileFormInput("myFile");
System.out.println(file.getTag());
```

El código anterior crea la siguiente salida:

```
<input type="file" name="myFile" />
```

Clase HiddenFormInput

La clase HiddenFormInput representa un tipo de entrada oculta de un formulario HTML.

El ejemplo de código siguiente muestra cómo se crea un objeto HiddenFormInput:

```
HiddenFormInput hidden = new HiddenFormInput("account", "123456");
System.out.println(hidden.getTag()):
```

El código anterior genera el código HTML siguiente:

<input type="hidden" name="account" value="123456" />

En una página HTML, el tipo de entrada oculta (HiddenInputType) no se visualiza. Tan solo envía la información (en este caso sería el número de cuenta) de regreso al servidor.

Clase ImageFormInput

La clase ImageFormInput representa un tipo de entrada de formulario HTML que es una imagen.

Mediante los métodos proporcionados se pueden recuperar y actualizar numerosos atributos de la clase ImageFormInput; por ejemplo, puede:

- Obtener o establecer el fuente
- Obtener o establecer la alineación
- Obtener o establecer la altura
- Obtener o establecer la anchura

El ejemplo de código siguiente muestra cómo se crea un objeto ImageFormInput:

```
ImageFormInput image = new ImageFormInput("myPicture", "myPicture.gif");
image.setAlignment(HTMLConstants.TOP);
image.setHeight(81);
image.setWidth(100);
```

El ejemplo de código anterior genera el código HTML siguiente:

```
<input type="image" name="MyPicture" src="myPicture.gif" align="top"
height="81" width="100" />
```

Clase ResetFormInput

La clase ResetFormInput representa un tipo de entrada de un formulario HTML que es un botón de restablecer.

El ejemplo de código siguiente muestra cómo se crea un objeto ResetFormInput:

```
        ResetFormInput reset = new ResetFormInput();
    reset.setValue("Restablecer");
    System.out.println(reset.getTag());
```

El ejemplo de código anterior genera el código HTML siguiente:

```
    <input type="reset" value="Restablecer"
/>
```

Clase SubmitFormInput

La clase SubmitFormInput representa un tipo de entrada de un formulario HTML que es un botón de someter.

El ejemplo de código siguiente muestra cómo se crea un objeto SubmitFormInput:

```
SubmitFormInput submit = new SubmitFormInput();
submit.setValue("Someter");
System.out.println(submit.getTag());
```

El ejemplo de código anterior genera la siguiente salida:

```
<input type="submit" value="Someter" />
```

Clase TextFormInput

La clase TextFormInput representa en un formulario HTML un tipo de entrada que es una sola línea de texto. La clase TextFormInput proporciona métodos que permiten obtener y establecer el número máximo de caracteres que un usuario puede entrar en el campo de texto.

El ejemplo siguiente muestra cómo se crea un objeto TextFormInput nuevo:

```
TextFormInput text = new TextFormInput("userID");
text.setSize(40);
System.out.println(text.getTag());
```

El ejemplo de código anterior genera el siguiente código HTML:

```
<input type="text" name="userID" size="40" />
```

Clase PasswordFormInput

La clase PasswordFormInput representa un tipo de campo de entrada de contraseña de un formulario HTML.

El ejemplo de código siguiente muestra cómo se crea un objeto PasswordFormInput nuevo:

```
PasswordFormInput pwd = new PasswordFormInput("password");
pwd.setSize(12);
System.out.println(pwd.getTag());
```

El ejemplo de código anterior genera el siguiente código HTML:

```
<input type="password" name="password" size="12" />
```

Clase RadioFormInput

La clase RadioFormInput representa un tipo de entrada de formulario HTML que es un botón de selección. Éste, en el momento de construirse, se puede inicializar como seleccionado.

Una serie de botones de selección que tengan un mismo nombre de control forman un grupo de botones de selección. La clase RadioFormInputGroup crea grupos de botones de selección. En un momento dado únicamente puede seleccionarse un solo botón del grupo. Además, en el momento de construir el grupo, se puede inicializar como seleccionado un botón concreto.

El ejemplo de código siguiente muestra cómo se crea un objeto RadioFormInput nuevo:

```
    RadioFormInput radio = new RadioFormInput("age", "twentysomething",
"Edad entre 20 y 29", true);
    System.out.println(radio.getTag());
```

El ejemplo de código anterior genera el código HTML siguiente:

```
    <input type="radio" name="age" value="twentysomething" checked="checked"
/>
```

Clase CheckboxFormInput

La clase CheckboxFormInput representa un tipo de entrada de formulario HTML que corresponde a un recuadro de selección. El usuario puede seleccionar más de una de las elecciones presentadas como recuadros de selección dentro de un formulario.

El ejemplo siguiente muestra cómo se crea un objeto CheckboxFormInput nuevo:

```
    CheckboxFormInput checkbox = new CheckboxFormInput("uscitizen", "yes",
"textLabel", true);
    System.out.println(checkbox.getTag());
```

El código anterior genera la siguiente salida:

```
    <input type="checkbox" name="uscitizen" value="yes" checked="checked" />
textLabel
```

Clase LayoutFormPanel

La clase LayoutFormPanel representa un diseño de los elementos de formulario de un formulario HTML. Puede utilizar los métodos proporcionados por la clase LayoutFormPanel para añadir o eliminar elementos de un panel o para obtener el número de elementos que hay en el diseño. Puede optar por utilizar uno de estos dos diseños:

- GridLayoutFormPanel: representa un diseño cuadriculado de los elementos de un formulario HTML.
- LineLayoutFormPanel: representa un diseño lineal de los elementos de un formulario HTML.

GridLayoutFormPanel

La clase GridLayoutFormPanel representa un diseño cuadriculado de los elementos de formulario.Este diseño se utiliza en un formulario HTML en el que se especifica el número de columnas de la cuadrícula.

En el ejemplo que hay a continuación se crea un objeto GridLayoutFormPanel con dos columnas:

```
        // Cree un elemento de entrada de formulario de texto para el
sistema.
  LabelFormElement sysPrompt = new LabelFormElement("Sistema:");
  TextFormInput system = new TextFormInput("System");


        // Cree un elemento de entrada de formulario de texto para el ID de
usuario.
  LabelFormElement userPrompt = new LabelFormElement("Usuario:");
  TextFormInput user = new TextFormInput("User");


        // Cree un elemento de entrada de formulario de contraseña para la
contraseña.
  LabelFormElement passwordPrompt = new LabelFormElement("Contraseña:");
  PasswordFormInput password = new PasswordFormInput("Password");


        // Cree el objeto GridLayoutFormPanel con dos columnas y añada los
elementos de formulario.
  GridLayoutFormPanel panel = new GridLayoutFormPanel(2);
  panel.addElement(sysPrompt);
  panel.addElement(system);
  panel.addElement(userPrompt);
  panel.addElement(user);
  panel.addElement(passwordPrompt);
  panel.addElement(password);


        // Cree el botón de someter para el formulario.
  SubmitFormInput logonButton = new SubmitFormInput("logon", "Iniciar
sesión");


        // Cree el objeto HTMLForm y añádale el panel.
  HTMLForm form = new HTMLForm(servletURI);
  form.addElement(panel);
  form.addElement(logonButton);
```

Este ejemplo genera el siguiente código HTML:

```
<form action=servletURI method="get">
<table border="0">
<tr>
<td>Sistema:</td>
<td><input type="text" name="System" /></td>
</tr>
<tr>
```

```
<td>Usuario:</td>
<td><input type="text" name="User" /></td>
</tr>
<tr>
<td>Contraseña:</td>
<td><input type="password" name="Password" /></td>
</tr>
</table>
<input type="submit" name="logon" value="Iniciar sesión" />
</form>
```

Clase LineLayoutFormPanel

La clase LineLayoutFormPanel representa un diseño lineal de los elementos de un formulario HTML. Los elementos del formulario se disponen en una sola fila dentro de un panel.

Este ejemplo crea un objeto LineLayoutFormPanel y añade dos elementos de formulario.

```
    CheckboxFormInput privacyCheckbox = new
CheckboxFormInput("confidential", "yes", "Confidencial", true);
    CheckboxFormInput mailCheckbox = new CheckboxFormInput("mailingList",
"yes", "Únase a nuestra lista de correo", false);
  LineLayoutFormPanel panel = new LineLayoutFormPanel();
  panel.addElement(privacyCheckbox);
  panel.addElement(mailCheckbox);
  String tag = panel.getTag();
```

El ejemplo de código anterior genera el siguiente código HTML:

```
    <input type="checkbox" name="confidential" value="yes"
    checked="checked" /> Confidencial <input type="checkbox"
    name="mailingList" value="yes" /> Únase a nuestra lista de correo <br/>
```

Clase TextAreaFormElement

La clase TextAreaFormElement representa un elemento de formulario HTML que es un área de texto. Para determinar el tamaño del área de texto, debe establecer el número de filas y columnas. Para averiguar qué tamaño se ha establecido para un área de texto, puede utilizar los métodos getRows() y getColumns().

Para establecer el texto inicial dentro del área de texto, se utiliza el método setText(). Utilice el método getText() para ver el texto inicial que se ha establecido.

El ejemplo siguiente muestra cómo se crea un objeto TextAreaFormElement:

```
TextAreaFormElement textArea = new TextAreaFormElement("foo", 3, 40);
textArea.setText("Aquí se pone el valor de TEXTAREA por omisión");
System.out.println(textArea.getTag());
```

El ejemplo de código anterior genera el siguiente código HTML:

```
<form>
<textarea name="foo" rows="3" cols="40">
Aquí se pone el valor de TEXTAREA por omisión
</textarea>
</form>
```

Clase LabelFormElement

La clase LabelFormElement representa una etiqueta para un elemento de formulario HTML. La clase LabelFormElement permite etiquetar elementos de un formulario HTML como, por ejemplo, un área de texto o una entrada de formulario de contraseña. La etiqueta es una línea de texto que se establece mediante el método setLabel(). Este texto no responde a la entrada de usuario y sirve para que al usuario le sea más fácil comprender el formulario.

El ejemplo de código siguiente muestra cómo se crea un objeto LabelFormElement:

```
LabelFormElement label = new LabelFormElement("Saldo de cuenta");
System.out.println(label.getTag());
```

Este ejemplo genera la siguiente salida:

```
Saldo de cuenta
```

Clase SelectFormElement

La clase SelectFormElement representa un tipo de entrada de selección de un formulario HTML. Puede añadir y eliminar diversas opciones dentro del elemento de selección.

La clase SelectFormElement dispone de métodos que le permiten ver y cambiar atributos del elemento de selección:

- Utilice setMultiple() para establecer si el usuario puede o no seleccionar más de una opción
- Utilice getOptionCount() para averiguar cuántos elementos hay en el diseño de la opción
- Utilice setSize() para establecer el número de opciones visibles dentro del elemento de selección, y el método getSize() para determinar el número de opciones visibles

En el ejemplo siguiente se crea un objeto SelectFormElement que tiene tres opciones. El objeto SelectFormElement que se llama *list* está resaltado. Las dos primeras opciones que se añaden especifican el texto de la opción, que es el nombre, y los atributos de selección. La tercera opción que se añade se define mediante un objeto SelectOption.

```
    SelectFormElement list = new SelectFormElement("list1");
    SelectOption option1 = list.addOption("Opción1", "opt1");
    SelectOption option2 = list.addOption("Opción2", "opt2", false);
    SelectOption option3 = new SelectOption("Opción3", "opt3", true);
  list.addOption(option3);
  System.out.println(list.getTag());
```

El ejemplo de código anterior genera el código HTML siguiente:

```
    <select name="list1">
    <option value="opt1">Opción1</option>
    <option value="opt2">Opción2</option>
    <option value="opt3" selected="selected">Opción3</option>
    </select>
```

Clase SelectOption

La clase SelectOption representa una opción de un elemento de formulario que es una opción HTML. El elemento de formulario opción se utiliza en un formulario de selección.

Se proporcionan métodos que permiten recuperar y establecer atributos dentro de una opción de selección (SelectOption). Por ejemplo, puede establecer si, por omisión, la opción debe estar o no seleccionada. También puede establecer el valor de entrada que la opción utilizará cuando se someta el formulario.

El siguiente ejemplo crea tres objetos SelectOption dentro de un formulario de selección. Todos los objetos SelectOption que figuran más abajo están resaltados. Se llaman *option1, option2* y *option3*. El objeto *option3* está inicialmente seleccionado.

```
    SelectFormElement list = new SelectFormElement("list1");
    SelectOption option1 = list.addOption("Opción1", "opt1");
    SelectOption option2 = list.addOption("Opción2", "opt2", false);
    SelectOption option3 = new SelectOption("Opción3", "opt3", true);
  list.addOption(option3);
  System.out.println(list.getTag());
```

El ejemplo de código anterior genera el código HTML siguiente:

```
    <select name="list1">
    <option value="opt1">Opción1</option>
    <option value="opt2">Opción2</option>
    <option value="opt3" selected="selected">Opción3</option>
    </select>
```

Clase RadioFormInputGroup

La clase RadioFormInputGroup representa un grupo de objetos RadioFormInput. Los usuarios solo pueden seleccionar uno de los objetos RadioFormInput de un RadioFormInputGroup.

Los métodos de la clase RadioFormInputGroup permiten trabajar con los diversos atributos de un grupo de botones de selección. Con estos métodos puede hacer las siguientes tareas:

- Añadir un botón de selección
- Eliminar un botón de selección
- Obtener o establecer el nombre del grupo de botones de selección

En el ejemplo que hay a continuación se crea un grupo de botones de selección:

```
      // Cree algunos botones de selección.
    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);
    // Cree un grupo de botones de selección y añada los botones de
selección.
    RadioFormInputGroup ageGroup = new RadioFormInputGroup("age");
  ageGroup.add(radio0);
  ageGroup.add(radio1);
  ageGroup.add(radio2);
  ageGroup.add(radio3);
  System.out.println(ageGroup.getTag());
```

El ejemplo de código anterior genera el siguiente código HTML:

```
    <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
```

Clase HTMLHeading

La clase HTMLHeading representa una cabecera HTML. Cada una de las cabeceras puede tener su propia alineación y su propio nivel de 1 (font más grande, mayor importancia) a 6.

Los métodos de la clase HTMLHeading son:

- Obtener y establecer el texto de la cabecera
- Obtener y establecer el nivel de la cabecera
- Obtener y establecer la alineación de la cabecera
- Obtener y establecer la dirección de interpretación del texto
- Obtener y establecer el idioma del elemento de entrada
- Obtener una representación de tipo String del objeto HTMLHeader

Ejemplo: crear objetos HTMLHeading

El ejemplo que sigue crea tres objetos HTMLHeading:

```
     // Cree y visualice tres objetos HTMLHeading.
     HTMLHeading h1 = new HTMLHeading(1, "Cabecera", HTMLConstants.LEFT);
     HTMLHeading h2 = new HTMLHeading(2, "Subcabecera",
HTMLConstants.CENTER);
     HTMLHeading h3 = new HTMLHeading(3, "Elemento", HTMLConstants.RIGHT);
     System.out.print(h1 + "\r\n" + h2 + "\r\n" + h3);
```

El ejemplo anterior genera los códigos siguientes:

```
     <h1 align="left">Cabecera</h1>
     <h2 align="center">Subcabecera</h2>
     <h3 align="right">Elemento</h3>
```

Clase HTMLHyperlink

La clase HTMLHyperlink representa un código de hiperenlace HTML. Con la clase HTMLHyperlink puede crear un enlace dentro de la página HTML. Esta clase le permite obtener y establecer numerosos atributos de hiperenlaces, como los que se indican a continuación:

- Obtener o establecer el URI (identificador de recursos uniforme) del enlace
- Obtener o establecer el título del enlace
- Obtener o establecer el marco destino del enlace

La clase HTMLHyperlink puede imprimir el hiperenlace completo con las propiedades definidas para poder utilizar la salida en la página HTML.

A continuación figura un ejemplo de HTMLHyperlink:

```
// Cree un hiperenlace HTML con la página de presentación de IBM Toolbox
para Java.
HTMLHyperlink toolbox = new
HTMLHyperlink("http://www.ibm.com/as400/toolbox", "Página de presentación de
IBM Toolbox para Java");

// Visualice el código de enlace de la Caja de Herramientas (Toolbox).
System.out.println(toolbox.toString());
```

El código anterior genera el código HTML siguiente:
Página de presentación de IBM Toolbox para Java

Este código, cuando se utiliza en una página HTML, ofrece este aspecto:

Página de presentación de IBM Toolbox para Java

»Clase HTMLImage

La clase HTMLImage permite crear códigos de imagen para la página HTML. La clase HTMLImage proporciona métodos que permiten obtener y establecer los atributos de imagen, tales como:

- Obtener o establecer la altura de la imagen
- Obtener o establecer la anchura de la imagen
- Obtener o establecer el nombre de la imagen
- Obtener o establecer el texto alternativo de la imagen
- Obtener o establecer el espacio horizontal alrededor de la imagen
- Obtener o establecer el espacio vertical alrededor de la imagen
- Obtener o establecer la referencia absoluta o relativa de la imagen
- Recuperar una representación de tipo String del objeto HTMLImage

El ejemplo siguiente muestra un modo de crear un objeto HTMLImage:

```
// Cree un objeto HTMLImage.
HTMLImage image = new HTMLImage("http://myWebSite/picture.gif",
                                "Texto alternativo para este gráfico");
image.setHeight(94);
image.setWidth(105);
System.out.println(image);
```

La sentencia de imprimir genera el siguiente código en una sola línea. La acomodación de texto sólo se produce para la visualización.

```
<img src="http://myWebSite/picture.gif"
     alt="Texto alternativo para este gráfico"
     height="94" width="105" />
```

«

Clases HTMLList

Las clases HTMLList permiten crear listas dentro de las páginas HTML con gran facilidad. Estas clases proporcionan métodos para obtener y establecer los diversos atributos de las listas y los elementos que contienen.

En concreto, la clase padre HTMLList proporciona un método para generar una lista compacta que visualiza los elementos en vertical en el menor espacio posible.

- Los métodos de HTMLList permiten:
  - Obtener una lista compacta
  - Añadir y eliminar elementos de la lista
  - Añadir y eliminar listas de la lista (permitiendo anidar listas)
- Los métodos de HTMLListItem permiten:
  - Obtener y establecer el contenido del elemento
  - Obtener y establecer la dirección de interpretación del texto
  - Obtener y establecer el idioma del elemento de entrada

Las subclases de HTMLList y HTMLListItem permiten crear listas HTML propias:

- OrderedList y OrderedListItem
- UnorderedList y UnorderedListItem

Para la codificación de snippets, consulte los ejemplos siguientes:

- **Ejemplo:** crear listas ordenadas
- **Ejemplo:** crear listas sin ordenar
- **Ejemplo:** crear listas anidadas

OrderedList y OrderedListItem

Las clases OrderedList y OrderedListItem permiten crear listas ordenadas en las páginas HTML.

- Los métodos de OrderedList permiten:
  - Obtener y establecer el número inicial del primer elemento de la lista
  - Obtener y establecer el tipo (o estilo) de los números de elemento
- Los métodos de OrderedListItem permiten:
  - Obtener y establecer el número del elemento
  - Obtener y establecer el tipo (o estilo) del número de elemento

Con los métodos de OrderedListItem, puede alterar temporalmente la numeración y el tipo de un elemento específico de la lista.

Vea el ejemplo para crear listas ordenadas.

UnorderedList y UnorderedListItem

Las clases UnorderedList y UnorderedListItem permiten crear listas sin ordenar en las páginas HTML.

- Los métodos de UnorderedList permiten:
  - Obtener y establecer el tipo (o estilo) de los elementos

- Los métodos de UnorderedListItem permiten:
  - Obtener y establecer el tipo (o estilo) del elemento

Vea el ejemplo para crear listas sin ordenar.

Ejemplos

Los ejemplos siguientes muestran cómo se utilizan las clases HTMLList para crear listas ordenadas, listas sin ordenar y listas anidadas.

Ejemplo: crear listas ordenadas

A continuación figura un ejemplo en el que se crea una lista ordenada:

```
     // Cree un objeto OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
     // Cree los objetos OrderedListItem.
OrderedListItem listItem1 = new OrderedListItem();
OrderedListItem listItem2 = new OrderedListItem();
     // Establezca los datos de los objetos OrderedListItem.
listItem1.setItemData(new HTMLText("Primer elemento"));
listItem2.setItemData(new HTMLText("Segundo elemento"));
     // Añada los elementos de lista al objeto OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
System.out.println(oList.getTag());
```

El ejemplo anterior genera los códigos siguientes:

```
   <ol type="i">
<li>Primer elemento</li>
<li>Segundo elemento</li>
   </ol>
```

Estos códigos, cuando se utilizan en una página HTML, ofrecen este aspecto:

i. Primer elemento
ii. Segundo elemento

Ejemplo: crear listas sin ordenar

A continuación figura un ejemplo en el que se crea una lista sin ordenar:

```
     // Cree un objeto UnorderedList.
UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
     // Cree los objetos UnorderedListItem.
UnorderedListItem listItem1 = new UnorderedListItem();
UnorderedListItem listItem2 = new UnorderedListItem();
     // Establezca los datos de los objetos UnorderedListItem.
listItem1.setItemData(new HTMLText("Primer elemento"));
listItem2.setItemData(new HTMLText("Segundo elemento"));
     // Añada los elementos de lista al objeto UnorderedList.
uList.addListItem(listItem1);
uList.addListItem(listItem2);
System.out.println(uList.getTag());
```

El ejemplo anterior genera los códigos siguientes:

```
<ul type="square">
<li>Primer elemento</li>
<li>Segundo elemento</li>
</ul>
```

Estos códigos, cuando se utilizan en una página HTML, ofrecen este aspecto:

- Primer elemento
- Segundo elemento

Ejemplo: crear listas anidadas

A continuación figura un ejemplo en el que se crea una lista anidada:

```
     // Cree un objeto UnorderedList.
UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
     // Cree y establezca los datos de los objetos UnorderedListItem.
UnorderedListItem listItem1 = new UnorderedListItem();
UnorderedListItem listItem2 = new UnorderedListItem();
listItem1.setItemData(new HTMLText("Primer elemento"));
listItem2.setItemData(new HTMLText("Segundo elemento"));
     // Añada los elementos de lista al objeto UnorderedList.
uList.addListItem(listItem1);
uList.addListItem(listItem2);

     // Cree un objeto OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
     // Cree los objetos OrderedListItem.
OrderedListItem listItem1 = new OrderedListItem();
OrderedListItem listItem2 = new OrderedListItem();
OrderedListItem listItem3 = new OrderedListItem();
     // Establezca los datos de los objetos OrderedListItem.
listItem1.setItemData(new HTMLText("Primer elemento"));
listItem2.setItemData(new HTMLText("Segundo elemento"));
listItem3.setItemData(new HTMLText("Tercer elemento"));
     // Añada los elementos de lista al objeto OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
     // Añada (anide) la lista sin ordenar a OrderedListItem2
oList.addList(uList);
     // Añada otro objeto OrderedListItem al objeto OrderedList
     // después del objeto UnorderedList anidado.
oList.addListItem(listItem3);
System.out.println(oList.getTag());
```

El ejemplo anterior genera los códigos siguientes:

```
   <ol type="i">
<li>Primer elemento</li>
<li>Segundo elemento</li>
   <ul type="square">
<li>Primer elemento</li>
<li>Segundo elemento</li>
</ul>
   <li>Tercer elemento</li>
   </ol>
```

Clase HTMLMeta

La clase HTMLMeta representa información de metadatos que se utiliza dentro de un código HTMLHead. Los atributos de los códigos META se emplean al identificar, indexar y definir información en el documento HTML.

Los atributos del código META son:

- NAME - nombre asociado al contenido del código META
- CONTENT - valores asociados al atributo NAME
- HTTP-EQUIV - información recopilada por servidores HTTP para las cabeceras de los mensajes de respuesta
- LANG - idioma
- URL - se utiliza para redirigir a los usuarios de la página actual a otro URL

Por ejemplo, para ayudar a los motores de búsqueda a determinar el contenido de una página, podría utilizar el código META que se indica a continuación:

```
<META name="keywords" lang="en-us" content="games, cards, bridge">
```

También puede emplear una clase HTMLMeta para redirigir a un usuario de una página a otra.

Los métodos de la clase HTMLMeta permiten:

- Obtener y establecer el atributo NAME
- Obtener y establecer el atributo CONTENT
- Obtener y establecer el atributo HTTP-EQUIV
- Obtener y establecer el atributo LANG
- Obtener y establecer el atributo URL

Ejemplo: crear códigos META

En el ejemplo siguiente se crean dos códigos META:

```
     // Cree un código META para ayudar a los motores de búsqueda a
determinar el contenido de la página.
     HTMLMeta meta1 = new HTMLMeta();
     meta1.setName("keywords");
     meta1.setLang("en-us");
     meta1.setContent("games, cards, bridge");
     // Cree un código META para que las antememorias puedan determinar
cuándo renovar la página.
     HTMLMeta meta2 = new HTMLMeta("Expires", "Mon, 01 Jun 2000 12:00:00
GMT");
     System.out.print(meta1 + "\r\n" + meta2);
```

El ejemplo anterior genera los códigos siguientes:

```
     <meta name="keywords" content="games, cards, bridge">
     <meta http-equiv="Expires" content="Mon, 01 Jun 2000 12:00:00 GMT">
```

Clase HTMLParameter

La clase HTMLParameter representa los parámetros que puede utilizar con la clase HTMLServlet. Cada uno de los parámetros tiene un nombre y un valor propios.

Los métodos de la clase HTMLParameter permiten:

- Obtener y establecer el nombre del parámetro
- Obtener y establecer el valor del parámetro

Ejemplo: crear códigos HTMLParameter

En el ejemplo siguiente se crea un código HTMLParameter:

```
     // Cree un objeto HTMLServletParameter.
HTMLParameter parm = new HTMLParameter ("age", "21");
System.out.println(parm);
```

El ejemplo anterior genera el código siguiente:

```
<param name="age" value="21">
```

Clase HTMLServlet

La clase HTMLServlet representa un elemento include en el lado del servidor. El objeto de servlet especifica el nombre del servlet y, de forma opcional, su ubicación. También puede elegir emplear la ubicación por omisión en el sistema local.

La clase HTMLServlet se utiliza con la clase HTMLParameter, que especifica los parámetros disponibles para el servlet.

Los métodos de la clase HTMLServlet permiten:

- Añadir y eliminar elementos HTMLParameter del código del servlet
- Obtener y establecer la ubicación del servlet
- Obtener y establecer el nombre del servlet
- Obtener y establecer el texto alternativo del servlet

Ejemplo: crear códigos HTMLServlet

El ejemplo siguiente añade un código HTMLServlet:

```
          // Cree un objeto HTMLServlet.
     HTMLServlet servlet = new HTMLServlet("myServlet",
"http://server:port/dir");
          // Cree un parámetro y, a continuación, añádalo al servlet.
     HTMLParameter param = new HTMLParameter("parm1", "value1");
     servlet.addParameter(param);
          // Cree y añada un segundo parámetro.
     HTMLParameter param2 = servlet.add("parm2", "value2");
          // Cree el texto alternativo si el servidor Web no soporta el
código de servlet.
     servlet.setText("El servidor Web que proporciona esta página no da
soporte al código SERVLET.")
     System.out.println(servlet);
```

El ejemplo anterior genera los códigos siguientes:

```
     <servlet name="myServlet" codebase="http://server:port/dir">
     <param name="parm1" value="value1">
     <param name="parm2" value="value2">
     El servidor Web que proporciona esta página no da soporte al código
SERVLET.
     </servlet>
```

Clases de tablas HTML

La clase HTMLTable permite preparar de manera sencilla las tablas que se pueden utilizar en las páginas HTML. Esta clase proporciona métodos para obtener y establecer los diversos atributos de la tabla, como se indica a continuación:

- Obtener y establecer la anchura del borde
- Obtener el número de filas de la tabla
- Añadir una columna o fila al final de la tabla
- Eliminar una columna o fila situada en una posición de columna o fila especificada

La clase HTMLTable emplea otras clases HTML para que resulte todavía más fácil crear una tabla. Las otras clases HTML que ayudan a crear tablas son:

- HTMLTableCell: crea una casilla de tabla
- HTMLTableRow: crea una fila de tabla
- HTMLTableHeader: crea una casilla de cabecera de tabla
- HTMLTableCaption: crea un pie de tabla

Ejemplo

Ejemplo: cómo se utilizan las clases HTMLTable.

Clase HTMLTableCell

La clase HTMLTableCell toma como entrada cualquier objeto HTMLTagElement y crea el código HTML de casilla de tabla con el elemento especificado. El elemento se puede establecer en el constructor o mediante uno de los dos métodos setElement().

Se pueden recuperar o actualizar muchos atributos de las casillas utilizando los métodos proporcionados en la clase HTMLTableCell. Algunas de las acciones que se pueden realizar con estos métodos son:

- Obtener o establecer el tramo de fila
- Obtener o establecer la altura de la casilla
- Establecer si los datos de la casilla van a utilizar o no los convenios de desglose normal de línea HTML

A continuación figura un ejemplo que crea un objeto HTMLTableCell y visualiza el código HTML:

```
//Cree un objeto HTMLHyperlink.
HTMLHyperlink link = new HTMLHyperlink("http://www.ibm.com",
                     "Página de presentación de IBM");
HTMLTableCell cell = new HTMLTableCell(link);
cell.setHorizontalAlignment(HTMLConstants.CENTER);
System.out.println(cell.getTag());
```

El método getTag() anterior proporciona la salida del ejemplo:

<td align="center">Página de presentación de IBM</td>

Clase HTMLTableRow

La clase HTMLTableRow crea una fila dentro de una tabla. Esta clase proporciona diversos métodos para obtener y establecer los atributos de una fila. Algunas de las tareas que se pueden realizar con estos métodos son:

- Añadir una columna a la fila o eliminar una columna de la fila
- Obtener datos de la columna que tiene el índice especificado
- Obtener el índice de la columna que tiene la casilla especificada
- Obtener el número de columnas que hay en una fila
- Establecer la alineación horizontal y vertical

A continuación figura un ejemplo de HTMLTableRow:

```
// Cree una fila y establezca la alineación.
HTMLTableRow row = new HTMLTableRow();
row.setHorizontalAlignment(HTMLTableRow.CENTER);

// Cree la información de columna y añádala a la fila.
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));

// Añada la fila a un objeto HTMLTable (se presupone que ya existe la
tabla).
table.addRow(row);
```

Clase HTMLTableHeader

La clase HTMLTableHeader es heredera de la clase HTMLTableCell. Crea un tipo específico de casilla, que es la casilla de cabecera, proporcionando una casilla de tipo **<th>**, en vez de una casilla de tipo **<td>**. Al igual que sucede con la clase HTMLTableCell, se puede llamar a diversos métodos con el fin de actualizar o recuperar los atributos de la casilla de cabecera.

A continuación figura un ejemplo de HTMLTableHeader:

```
// Cree las cabeceras de la tabla.
HTMLTableHeader account_header = new HTMLTableHeader(new
HTMLText("CUENTA"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NOMBRE"));
HTMLTableHeader balance_header = new HTMLTableHeader();
HTMLText balance = new HTMLText("SALDO");
balance_header.setElement(balance);

// Añada las cabeceras de tabla a un objeto HTMLTable (se presupone que ya
existe la tabla).
table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);
```

Clase HTMLTableCaption

La clase HTMLTableCaption crea un pie para la tabla HTML. Proporciona métodos para actualizar y recuperar los atributos del pie de tabla. Por ejemplo, puede utilizar setAlignment() para especificar con qué componente de la tabla se debe alinear el pie de tabla. A continuación figura un ejemplo de HTMLTableCaption:

```
// Cree un objeto HTMLTableCaption por omisión y establezca el texto del pie
de tabla.
HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Saldos de cuenta de cliente - 1 de enero de 2000");

// Añada el pie de tabla a un objeto HTMLTable (se presupone que ya existe
la tabla).
table.setCaption(caption);
```

Clase HTMLText

La clase HTMLText permite acceder a las propiedades del texto de la página HTML. Mediante la clase HTMLText puede obtener, establecer y comprobar el estado de numerosos atributos del texto, como los que se indican a continuación:

- Obtener o establecer el tamaño del font
- Activar o desactivar el atributo de negrita (true o false) o averiguar si ya está activo
- Activar o desactivar el atributo de subrayado (true o false) o averiguar si ya está activo
- Obtener o establecer la alineación horizontal del texto

A continuación figura un ejemplo en el que se crea un objeto HTMLText, se activa el atributo de negrita y se establece el tamaño del font en 5.

```
HTMLText text = new HTMLText("IBM");
text.setBold(true);
text.setSize(5);
System.out.println(text.getTag());
```

La sentencia de imprimir genera el siguiente código HTML:

```
<font size="5"><b>IBM</b></font>
```

Este código, cuando se utiliza en una página HTML, ofrece este aspecto:

IBM

Clases HTMLTree

La clase HTMLTree permite preparar de manera sencilla un árbol jerárquico de elementos HTML que se puede utilizar en las páginas HTML. Esta clase proporciona métodos para obtener y establecer los diversos atributos del árbol, además de los métodos que permiten llevar a cabo las acciones siguientes:

- Obtener y establecer la petición de servlet HTTP
- Añadir un objeto HTMLTreeElement o FileTreeElement al árbol
- Eliminar un objeto HTMLTreeElement o FileTreeElement del árbol

La clase HTMLTree utiliza otras clases HTML que facilitan la creación de un árbol jerárquico:

- HTMLTreeElement: crea un elemento de árbol
- FileTreeElement: crea un elemento de árbol de archivo
- FileListElement: crea un elemento de lista de archivos
- »FileListRenderer: representa la lista de archivos y directorios«

Ejemplos

En los ejemplos siguientes se muestran diversas formas de utilizar las clases HTMLTree.

- **Ejemplo:** cómo se utilizan las clases HTMLTree
- **Ejemplo:** crear un árbol de sistema de archivos integrado que se pueda recorrer

Clase HTMLTreeElement

La clase HTMLTreeElement representa un elemento jerárquico dentro de un objeto HTMLTree o de otros objetos HTMLTreeElement.

Se pueden recuperar o actualizar muchos atributos de elementos de árbol utilizando los métodos proporcionados en la clase HTMLTreeElement. Algunas de las acciones que se pueden realizar con estos métodos son:

- Obtener o establecer el texto visible del elemento de árbol
- Obtener o establecer el URL del icono expandido y contraído
- Establecer si el elemento de árbol se expandirá o no

A continuación figura un ejemplo que crea un objeto HTMLTreeElement y visualiza el código HTML:

```
        // Cree un objeto HTMLTree.
        HTMLTree tree = new HTMLTree();

        // Cree el objeto HTMLTreeElement padre.
        HTMLTreeElement parentElement = new HTMLTreeElement();
        parentElement.setTextUrl(new HTMLHyperlink("http://MiPáginaWeb", "Mi
página Web"));


        // Cree el objeto HTMLTreeElement hijo.
        HTMLTreeElement childElement = new HTMLTreeElement();
        childElement.setTextUrl(new HTMLHyperlink("http://OtraPáginaWeb",
"Otra página Web"));
        parentElement.addElement(childElement);

        // Añada el elemento de árbol al árbol.
        tree.addElement(parentElement);
        System.out.println(tree.getTag());
```

El método getTag() del ejemplo anterior genera códigos HTML como los siguientes:

```
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>Mi página Web</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>Otra página Web</u></font> </td>
</tr>
</table>
</td>
</tr>
</table>
```

Clase FileTreeElement

La clase FileTreeElement representa el sistema de archivos integrado dentro de una vista HTMLTree.

Se pueden recuperar o actualizar muchos atributos de elementos de árbol utilizando los métodos proporcionados en la clase HTMLTreeElement. »También puede obtener y establecer el nombre y la vía de acceso de las unidades compartidas NetServer.«

Algunas de las acciones que estos métodos permiten llevar a cabo son:

- Obtener o establecer el URL del icono expandido y contraído (método heredado)
- Establecer si el elemento de árbol se expandirá o no (método heredado)
- »Obtener o establecer el nombre de la unidad compartida NetServer«
- »Obtener o establecer la vía de acceso de la unidad compartida NetServer«

A continuación figura un ejemplo que crea un objeto FileTreeElement y visualiza el código HTML:

```
        // Cree un objeto HTMLTree.
        HTMLTree tree = new HTMLTree();

           // Cree un objeto URLParser.
        URLParser urlParser = new
URLParser(httpServletRequest.getRequestURI());

    // Cree un objeto AS400.
       AS400 system = new AS400(mySystem, myUserId, myPassword);

      // Cree un objeto IFSJavaFile.
       IFSJavaFile root = new IFSJavaFile(system, "/QIBM");

       // Cree un objeto DirFilter y obtenga los directorios.
          DirFilter filter = new DirFilter();
       File[] dirList = root.listFiles(filter);

       for (int i=0; i < dirList.length; i++)
       {

           // Cree un objeto FileTreeElement.
            FileTreeElement node = new FileTreeElement(dirList[i]);

           // Establezca el URL del icono.
            ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
            sl.setHttpServletResponse(resp);
           element.setIconUrl(sl);

           // Añada el objeto FileTreeElement al árbol.
           tree.addElement(element);
       }

       System.out.println(tree.getTag());
```

El método getTag() anterior proporciona la salida del ejemplo.

Clase FileListElement

La clase FileListElement permite crear un elemento de lista de archivos, que representa el contenido de un directorio de sistema de archivos integrado.

»Puede utilizar el objeto FileListElement para representar el contenido de una unidad compartida NetServer obteniendo y estableciendo el nombre y la vía de acceso de las unidades compartidas NetServer.«

La clase FileListElement proporciona métodos que permiten llevar a cabo estas acciones:

- Listar y ordenar los elementos de la lista de archivos
- Obtener y establecer la petición de servlet HTTP
- Obtener y establecer el objeto FileListRenderer
- Obtener y establecer el objeto HTMLTable con el que se visualizará la lista de archivos
- »Obtener o establecer el nombre de una unidad compartida NetServer«
- »Obtener o establecer la vía de acceso de una unidad compartida NetServer«

Puede utilizar la clase FileListElement con otras clases del paquete html:

- Con una clase FileListRenderer, puede especificar cómo desea visualizar la lista de archivos.
- Con la clase FileTreeElement, puede crear una lista que se pueda recorrer de archivos de sistema de archivos integrado »o archivos compartidos NetServer«

El javadoc de FileListElement muestra cómo crear y visualizar un objeto FileListElement.

Ejemplo

El ejemplo que sigue muestra cómo se puede utilizar la clase FileListElement con las clases HTMLTree (FileTreeElement y HTMLTreeElement) para crear un árbol de sistema de archivos integrado que se pueda recorrer. »El ejemplo también contiene código para establecer la vía de acceso de una unidad compartida NetServer.«

- **Ejemplo:** crear un árbol de sistema de archivos integrado que se pueda recorrer

»Clase FileListRenderer

La clase FileListRenderer representa cualquier campo de los objetos File (directorios y archivos) en un objeto FileListElement.

La clase FileListRenderer ofrece métodos que permiten llevar a cabo las acciones siguientes:

- Obtener el nombre del directorio
- Obtener el nombre del archivo
- Obtener el nombre del directorio padre
- Devolver la fila de datos que desea visualizar en el objeto FileListElement

En este ejemplo se crea un objeto FileListElement con un representador:

```
// Cree un objeto FileListElement.
FileListElement fileList = new FileListElement(sys, httpservletrequest);

// Establezca el representador específico para este servlet, que amplía
// FileListRenderer y altera temporalmente los métodos aplicables.
fileList.setRenderer(new myFileListRenderer(request));
```

Si no desea utilizar el representador por omisión, puede ampliar FileListRenderer y alterar temporalmente los métodos o crear otros nuevos. Por ejemplo, puede desear asegurarse de impedir que se pasen los nombres de directorios o archivos específicos con unas extensiones determinadas al objeto FileListElement. Ampliando la clase y alterando temporalmente el método correspondiente, puede devolver un valor nulo para estos archivos y directorios, con lo que se asegura de que no se visualicen.

Para personalizar por completo las filas de un objeto FileListElement, utilice el método getRowData(). Un ejemplo de personalización de datos de filas mediante getRowData() sería añadir una columna a los datos de fila o reorganizar las columnas. Cuando el comportamiento por omisión del objeto FileListRenderer sea satisfactorio, no será necesario que lleve a cabo ninguna programación adicional ya que la clase FileListElement crea un objeto FileListRenderer por omisión.«

Clases ReportWriter

El paquete com.ibm.as400.util.reportwriter proporciona clases que permiten utilizar el iSeries para acceder a datos de un archivo fuente XML o datos generados por servlets o JavaServer Pages(TM) y darles formato con más facilidad. El paquete reportwriter ofrece un modo cómodo de especificar tres paquetes distintos pero relacionados:

- com.ibm.as400.util.reportwriter.pclwriter
- com.ibm.as400.util.reportwriter.pdfwriter
- com.ibm.as400.util.reportwriter.processor

»Estos paquetes contienen diversas clases que permiten dar formato a corrientes de datos XML y generar informes en esos formatos. Compruebe que tiene los archivos jar necesarios en la CLASSPATH. Para obtener más información acerca de los archivos jar de reportwriter, consulte Archivos jar.«

Las clases Context (de los paquetes pclwriter y pdfwriter) definen métodos que las clases ReportProcessor necesitan para representar datos XML y JSP en el formato elegido:

- Utilice PCLContext junto con una clase ReportWriter para generar un informe en el formato PCL (Printer Control Language) de Hewlett Packard.
- Utilice PDFContext junto con una clase ReportWriter para generar un informe en el formato PDF (Portable Document Format) de Adobe.

Las clases ReportProcessor (del paquete processor) permiten generar informes con formato a partir de la información que la aplicación recoge de los datos fuente XML, servlets Java y JavaServer Pages (JSP).

- Utilice la clase JSPReportProcessor para recuperar datos de servlets y páginas JSP para generar informes en los formatos disponibles (contextos).
- Utilice la clase XSLReportProcessor para procesar los datos XML con hojas de estilo XSL a fin de generar informes en los formatos disponibles (contextos).

Clases Context

Las clases Context soportan formatos de datos específicos que, en combinación con las clases OutputQueue y SpooledFileOutputStream, permiten a las clases ReportWriter generar informes en ese formato y colocar esos informes en un archivo en spool.

La aplicación sólo tiene que crear una instancia de la clase Context, que las clases ReportWriter posteriormente utilizan para generar los informes. La aplicación no debe nunca llamar directamente a ninguno de los métodos de ninguna clase Context. Los métodos PCLContext y PDFContext están pensados para el uso interno de las clases ReportWriter.

Para construir una instancia de la clase Context se necesita un OutputStream (del paquete java.io) y un PageFormat (del paquete java.awt.print). Los ejemplos siguientes muestran cómo es posible construir y utilizar las clases Context con otras clases ReportWriter para generar informes:

Ejemplo: cómo se utiliza XSLReportProcessor con PCLContext

Ejemplo: cómo se utiliza JSPReportProcessor con PDFContext

Clase JSPReportProcessor

La clase JSPReportProcessor permite crear un documento o informe a partir del contenido de un servlet Java o JavaServer Page(TM) (JSP).

Utilice esta clase para obtener un servlet o JSP de un URL determinado y crear un documento a partir del contenido. El servlet o JSP debe proporcionar los datos del documento, incluidos los objetos de formato XSL. El usuario debe especificar el contexto de salida y el origen de datos de entrada de JSP antes de poder generar las páginas del documento. A continuación puede convertir los datos del informe en un formato de corriente de datos de salida específico.

La clase JSPReportProcessor permite llevar a cabo estas acciones:

- Procesar el informe
- Establecer un URL como plantilla

Los ejemplos que hay a continuación muestran cómo pueden utilizarse las clases JSPReportProcessor y PDFContext para generar un informe. »Los ejemplos contienen el código Java y JSP, que puede ver mediante los enlaces siguientes. También puede bajar un archivo zip con los archivos fuente JSP, XML y XSL de ejemplo para los ejemplos de JSPReportProcessor y XSLReportProcessor:«

- Ejemplo: cómo se utiliza JSPReportProcessor con PDFContext
- »Ejemplo: archivo JSP de ejemplo de JSPReportProcessor«

Para obtener más información sobre JSP, consulte la información acerca de la tecnología de Java Server Pages en el sitio Web de Java de Sun.

Clase XSLReportProcessor

La clase XSLReportProcessor permite crear un documento o informe transformando los datos fuente XML y dándoles formato con una hoja de estilo XSL. Utilice esta clase para crear el informe empleando una hoja de estilo XSL que contenga los objetos de formato (FO) XSL, que deben cumplir la especificación XSL. A continuación utilice una clase Context para convertir los datos del informe en un formato de corriente de datos de salida específico.

La clase XSLReportProcessor permite llevar a cabo estas acciones:

- Establecer la hoja de estilo XSL
- Establecer el origen de datos XML
- Establecer el fuente FO XSL
- Procesar un informe

El ejemplo que sigue muestra cómo se pueden utilizar las clases XSLReportProcessor y PCLContext para generar un informe. »Los ejemplos contienen el código Java, XML y XSL, que puede ver mediante los enlaces siguientes. También puede bajar un archivo zip con los archivos fuente XML, XSL y JSP de ejemplo para los ejemplos de XSLReportProcessor y JSPReportProcessor: «

- Ejemplo: cómo se utiliza XSLReportProcessor con PCLContext
- »Ejemplo: archivo XML de ejemplo de XSLReportProcessor«
- »Ejemplo: archivo XSL de ejemplo de XSLReportProcessor«

Para obtener más información sobre XML y XSL, consulte el tema acerca de XML que se encuentra en Information Center.

Clases de recursos

El paquete com.ibm.as400.resource proporciona una infraestructura genérica para trabajar con diversas listas y objetos AS400. Esta infraestructura proporciona una interfaz de programación coherente para todos estos objetos y listas.

El paquete de recursos incluye las clases siguientes:

- Resource: objeto que representa un recurso de iSeries, como por ejemplo un usuario, una impresora, un trabajo, un mensaje o un archivo. Las subclases concretas de Resource son:
    - RIFSFile
    - RJavaProgram
    - RJob
    - RPrinter
    - RQueuedMessage
    - RSoftwareResource
    - RUser

  Nota: las clases de NetServer del paquete access también son subclases concretas de Resource.
- ResourceList: objeto que representa una lista de recursos de iSeries, como por ejemplo una lista de usuarios, impresoras, trabajos, mensajes o archivos. Las subclases concretas de Resource son:
    - RIFSFileList
    - RJobList
    - RJobLog
    - RMessageQueue
    - RPrinterList
    - RUserList
- Presentation: objeto que permite presentar información sobre objetos de recurso, listas de recursos, atributos, selecciones y ordenaciones a los usuarios finales.

Clases Resource y ChangeableResource

Las clases abstractas com.ibm.as400.resource.Resource y com.ibm.as400.resource.ChangeableResource representan un recurso de iSeries.

Resource

Resource es una clase abstracta que proporciona acceso genérico a los atributos de un recurso cualquiera. Cada uno de los atributos se identifica mediante un ID de atributo y toda subclase específica de Resource normalmente documentará los ID de atributo que admite.

Resource proporciona únicamente acceso de lectura a los valores de atributo.

IBM Toolbox para Java proporciona los siguientes objetos de recurso:

- RIFSFile: representa un archivo o directorio del sistema de archivos integrado de iSeries.
- RJavaProgram: representa un programa Java en iSeries.
- RJob: representa un trabajo de iSeries.
- RPrinter: representa una impresora de iSeries.
- RQueuedMessage: representa un mensaje en una cola de mensajes o en unas anotaciones de trabajo de iSeries.
- RSoftwareResource: representa un programa bajo licencia en iSeries.
- RUser: representa un usuario de iSeries.

ChangeableResource

La clase abstracta ChangeableResource, una subclase de Resource, añade la posibilidad de cambiar los valores de atributo de un recurso de iSeries. Los cambios en los atributos se almacenan internamente en la antememoria hasta que se comprometen o cancelan. Esto permite cambiar muchos valores de atributo de una sola vez.

Nota: las clases de NetServer del paquete access también son subclases concretas de Resource y ChangeableResource.

Ejemplos

Los ejemplos siguientes muestran cómo se pueden utilizar directamente las subclases concretas de Resource y ChangeableResource, además de cómo puede funcionar el código genérico con cualquier subclase de Resource o ChangeableResource.

- Recuperar un valor de atributo de RUser, una subclase concreta de Resource
- Establecer valores de atributo para RJob, una subclase concreta de ChangeableResource
- Cómo se utiliza el código genérico para acceder a los recursos

Listas de recursos

La clase com.ibm.as400.resource.ResourceList representa una lista de recursos de iSeries. Es una clase abstracta que proporciona acceso genérico al contenido de la lista.

IBM Toolbox para Java proporciona las siguientes listas de recursos:

- RIFSFileList: representa una lista de archivos y directorios en el sistema de archivos integrado de iSeries
- RJobList: representa una lista de trabajos de iSeries
- RJobLog: representa una lista de mensajes en unas anotaciones de trabajo de iSeries
- RMessageQueue: representa una lista de mensajes en una cola de mensajes de iSeries
- RPrinterList: representa una lista de impresoras de iSeries
- RUserList: representa una lista de usuarios de iSeries

Una lista de recursos siempre está abierta o cerrada. La lista de recursos debe estar abierta para que se pueda acceder a su contenido. Con objeto de proporcionar acceso inmediato al contenido de la lista y gestionar la memoria de forma eficaz, la mayoría de las listas de recursos se cargan de forma incremental.

Las listas de recursos permiten realizar estas tareas:

- Abrir la lista
- Cerrar la lista
- Acceder a un recurso específico de la lista
- Esperar a que se cargue un recurso determinado
- Esperar a que se cargue la lista de recursos completa

También puede filtrar las listas de recursos mediante valores de selección. Cada valor de selección se identifica mediante un ID de selección. De forma parecida, las listas de recursos pueden ordenarse mediante valores de ordenación. Cada valor de ordenación se identifica mediante un ID de ordenación. Normalmente toda subclase de ResourceList documentará los ID de selección y los ID de ordenación que admite.

Ejemplos

Los ejemplos siguientes muestran diversas formas de trabajar con listas de recursos:

- Ejemplo: obtener e imprimir el contenido de un objeto ResourceList
- Ejemplo: cómo se utiliza el código genérico para acceder a un objeto ResourceList
- Ejemplo: visualizar una lista de recursos en un servlet (tabla HTML)

Clase Presentation

Todos los objetos de recursos, listas de recursos y objetos de metadatos tienen asociado un objeto com.ibm.as400.resource.Presentation que proporciona información traducida, como por ejemplo el nombre, el nombre completo y el icono.

Ejemplo: imprimir una lista de recursos y sus valores de ordenación utilizando sus presentaciones

Puede utilizar la información de Presentation para visualizar información sobre objetos de recurso, listas de recursos, atributos, selecciones y ordenaciones a los usuarios finales en formato de texto.

```
   void printCurrentSort(ResourceList resourceList) throws ResourceException
   {
       // Obtenga la presentación del objeto ResourceList e imprima su
nombre completo.
       Presentation resourceListPresentation =
resourceList.getPresentation();
       System.out.println(resourceListPresentation.getFullName());

       // Obtenga el valor de ordenación actual.
       Object[] sortIDs = resourceList.getSortValue();

       // Imprima cada uno de los ID de ordenación.
       for(int i = 0; i < sortIDs.length; ++i)
       {
           ResourceMetaData sortMetaData =
resourceList.getSortMetaData(sortIDs[i]);
           System.out.println("Se está ordenando por " +
sortMetaData.getName());
       }
   }
```

Clases de seguridad

Utilice las clases de seguridad de IBM Toolbox para Java para proporcionar conexiones protegidas con un servidor, verificar la identidad de un usuario y asociar un usuario a la hebra de sistema operativo cuando se esté ejecutando en el servidor local. Los servicios de seguridad incluidos son:

- Java Secure Socket Extension (JSSE) proporciona conexiones seguras tanto mediante el cifrado de los datos intercambiados entre un cliente y una sesión de servidor como mediante la autenticación de servidor.

  Nota: la información acerca del uso de SSL (capa de sockets segura) sólo se facilita a efectos de compatibilidad con versiones anteriores.
- Los servicios de autenticación proporcionan la posibilidad de:
  - Autenticar una identidad de usuario y su contraseña en relación con el registro de usuarios de OS/400.
  - Asignar una identidad a la hebra de OS/400 actual.

SSL (capa de sockets segura)

SSL (capa de sockets segura) proporciona conexiones seguras mediante:

- El cifrado de los datos intercambiados entre un cliente y una sesión
- La autenticación de servidor

Nota: puede emplear Java Secure Socket Extension (JSSE) en lugar de los siguientes métodos para proporcionar conexiones seguras. La información siguiente acerca del uso de SSL sólo se facilita a efectos de compatibilidad con versiones anteriores.

La utilización de SSL incide de forma negativa en el rendimiento ya que las conexiones SSL son más lentas que las que no tienen cifrado. Utilice conexiones con SSL cuando la confidencialidad de los datos transferidos sea más importante que la disminución del rendimiento (por ejemplo, al transferir información de tarjetas de crédito o de estado de cuentas bancarias).

Antes de empezar a utilizar SSL con IBM Toolbox para Java, debe saber cuáles son sus responsabilidades legales.

Algoritmos de SSL

IBM Toolbox para Java no contiene los algoritmos necesarios para cifrar y descifrar datos. En la versión V5R2, estos algoritmos se distribuyen con el programa bajo licencia iSeries Client Encryption (128 bits), 5722-CE3.

Nota: Toolbox para Java también es compatible con el programa bajo licencia iSeries Client Encryption (56 bits), 5722-CE2, que ya no se actualiza y no está disponible para la versión V5R2. Dado que Client Encryption (56 bits) contiene algoritmos de menos fiabilidad que Client Encryption (128 bits), plantéese la posibilidad de actualizar al cifrado de 128 bits.

Póngase en contacto con el representante de IBM para obtener más información o hacer un pedido de Client Encryption (128 bits), 5722-CE3.

Puesta a punto del entorno de SSL

IBM Toolbox para Java admite dos entornos de utilización de SSL para el cifrado de datos que debe poner a punto debidamente.

- Utilización del cifrado entre las clases de IBM Toolbox para Java y los servidores OS/400
- Utilización del cifrado entre el cliente proxy y el servidor proxy

Compatibilidad con versiones anteriores de IBM Toolbox para Java

La versión V5R2 de IBM Toolbox para Java, los algoritmos de cifrado y los archivos de clase de archivo de claves requieren que se utilice la versión V5R1 o V5R2 de los programas bajo licencia Client Encryption.

Nota: si lleva a cabo una actualización de OS/400 Versión V4R5 o anterior, debe actualizar el archivo KeyRing.class.

Si utiliza IBM Toolbox para Java V5R2 y una versión compatible de Client Encryption en el cliente, puede conectarse a la versión V4R4 y posteriores de OS/400. Para obtener más información sobre las versiones compatibles de Client Encryption, consulte los algoritmos de SSL.

SSL - Responsabilidades legales

El producto bajo licencia IBM iSeries Client Encryption (128 bits) proporciona soporte de cifrado SSL Versión 3.0 utilizando algoritmos de cifrado de 128 bits.

Este programa contiene una tecnología de cifrado de datos que está sujeta a requisitos especiales de licencia de exportación del Departamento de Comercio de Estados Unidos. Otros países u otras regiones también pueden tener requisitos de licencia de importación y exportación.

Mediante este documento se le notifica que el uso (o la transferencia) del mismo programa por parte de los usuarios de un mismo país o una misma región o de distintos países o distintas regiones puede estar prohibido o sujeto a:

- Leyes, normas o políticas especiales de importación del gobierno nacional del usuario.
- Leyes, normas o políticas especiales de exportación de su gobierno nacional.

Usted acepta todas las responsabilidades con objeto de asegurar que el programa se utilice o se transfiera de acuerdo con todas las leyes, normas y políticas de importación y de exportación aplicables desde este momento y posteriormente a la caducidad de la licencia.

Usted y sus usuarios deben cumplir las leyes de importación/exportación de los otros países o las otras regiones.

Utilización de SSL para cifrar datos entre IBM Toolbox para Java y los servidores OS/400

Puede emplear SSL para cifrar los datos que se intercambian entre las clases de IBM Toolbox para Java y los servidores OS/400. En el lado del cliente, los archivos que se distribuyen con el programa bajo licencia IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) se utilizan para cifrar los datos. En el lado del servidor, debe utilizar el gestor de certificados digitales de OS/400 a fin de configurar los servidores OS/400 para intercambiar los datos cifrados.

Puesta a punto del cliente y del servidor para utilizar SSL

Para cifrar los datos que fluyen entre las clases de IBM Toolbox para Java y los servidores OS/400, lleve a cabo estas tareas:

1. Ponga a punto los servidores para intercambiar datos cifrados.
2. Ponga a punto el cliente (las clases de IBM Toolbox para Java) para intercambiar datos cifrados. El procedimiento para ello depende del tipo de certificado utilizado al poner a punto SSL en el servidor:
   - Utilización de un certificado de servidor de una autoridad de confianza
   - Utilización de un certificado con autofirma

     Nota: la puesta a punto del cliente utilizando un certificado de una autoridad de confianza es notablemente más sencilla y rápida que utilizando un certificado con autofirma.
3. Utilice el objeto SecureAS400 para forzar a IBM Toolbox para Java a cifrar datos.

   Nota: con la realización de los dos primeros pasos anteriores sólo se crea una vía de acceso segura entre el cliente y el servidor. La aplicación debe utilizar el objeto SecureAS400 para indicar a IBM Toolbox para Java qué datos debe cifrar. Los datos que fluyen por el objeto SecureAS400 son los únicos datos que se cifran. Si emplea un objeto AS400, los datos no se cifrarán y se utilizará la vía de acceso normal hasta el servidor.

Puesta a punto de los servidores iSeries para utilizar SSL

Para poner a punto los servidores iSeries para utilizar SSL con IBM Toolbox para Java, siga estos pasos:

1. »Instale lo siguiente en los servidores iSeries:
   - IBM Cryptographic Access Provider de 128 bits para iSeries, 5722-AC3, que proporciona cifrado en el lado del servidor.
   - iSeries Client Encryption (128 bits), 5722-CE3, que proporciona los programas de utilidad y las clases Java que las clases de IBM Toolbox para Java utilizan en el lado del cliente.

   Nota: Toolbox para Java también es compatible con la versión V5R1 de Cryptographic Access Provider de 56 bits para iSeries, 5722-AC2, y la versión V5R1 de Client Encryption (56 bits), 5722-CE2.«
2. Cambie la autorización del directorio que contiene los archivos de cifrado del cliente.
3. Obtenga y configure el certificado de servidor.
4. Aplique el certificado a los siguientes servidores iSeries utilizados por IBM Toolbox para 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

Cambiar la autorización del directorio que contiene los archivos de cifrado del cliente

Para ayudarle a cumplir las responsabilidades legales en relación con SSL requeridas al utilizar los algoritmos criptográficos, el directorio que contiene los archivos se distribuye con la autorización de uso público *EXCLUDE. Debe cambiar la autorización del directorio con objeto de permitir el acceso únicamente a los usuarios autorizados para utilizar los algoritmos de cifrado.

Utilice la seguridad de objetos de OS/400 para controlar el acceso a los archivos de cifrado del cliente con los pasos siguientes:

1. En el servidor, escriba el mandato siguiente:

   ```
   wrklnk '/QIBM/ProdData/HTTP/Public/jt400/*'
   ```
2. Seleccione la opción 9 en el directorio SSL56 o SSL128.
3. Asegúrese de que la autorización de tipo *PUBLIC es *EXCLUDE
4. A los usuarios o grupos de usuarios que necesiten acceder a los archivos SSL, otórgueles la autorización *RX sobre el directorio.

   Nota: no puede denegar el acceso a los archivos SSL a los usuarios que tengan la autorización especial *ALLOBJ.

Obtener y configurar certificados de servidor

Antes de obtener y configurar el certificado de servidor, debe instalar los productos siguientes:

- Programa bajo licencia IBM HTTP Server para iSeries (5722-DG1)
- Opción 34 del sistema operativo base (Gestor de Certificados Digitales)

El proceso que siga para obtener y configurar el certificado de servidor depende del tipo de certificado que utilice:

- Si obtiene un certificado de una autoridad de confianza (como por ejemplo VeriSign, Inc., o RSA Data Security, Inc.), instale el certificado en el iSeries y a continuación aplíquelo a los servidores de sistema principal.
- Si opta por no utilizar un certificado de una autoridad de confianza, puede construir su propio certificado para ser utilizado en el iSeries. Construya el certificado mediante el Gestor de Certificados Digitales.

    1. Cree la autoridad certificadora en el servidor iSeries. Consulte el tema correspondiente del Information Center, Acting as your own CA.
    2. Cree un certificado del sistema a partir de la autoridad certificadora que ha creado.
    3. Asigne qué servidores de sistema principal van a utilizar el certificado del sistema que ha creado.

Utilización de un certificado de una autoridad de confianza

Junto con IBM Toolbox para Java se proporciona un archivo de claves que da soporte a certificados de servidor procedentes de un conjunto de autoridades de confianza, representadas por las compañías siguientes:

- IBM World Registry
- Integrion Financial Network
- RSA Data Security, Inc.
- Thawte Consulting
- VeriSign, Inc.

El archivo de claves ya soporta los certificados que se obtienen de una de estas autoridades de confianza. Tan solo debe obtener los archivos zip que contienen los algoritmos de cifrado y añadirlos a la sentencia CLASSPATH.

Para utilizar el certificado, siga estos pasos:

1. Seleccione el directorio en que desea colocar los archivos zip.
2. Baje la versión de SSL que desee utilizar copiando los archivos zip en el directorio seleccionado:
    - Para el cifrado de 56 bits (utilizado con el programa bajo licencia 5722-CE2), copie /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip.
    - Para el cifrado de 128 bits (utilizado con el programa bajo licencia 5722-CE3), copie /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip.
3. Añada el archivo zip a la sentencia CLASSPATH.

Utilización de un certificado con autofirma

»Si elige no utilizar un certificado de una autoridad de confianza, debe bajar el certificado de CA con autofirma (de cada uno de los servidores que tiene un certificado de CA con autofirma) para que las clases de IBM Toolbox para Java puedan utilizarlo.« También debe obtener los archivos zip que contienen los algoritmos de cifrado y añadirlos a la sentencia CLASSPATH.

Para utilizar el certificado con autofirma, siga estos pasos:

1. Seleccione el directorio en que desea colocar los archivos zip.
2. Baje la versión de SSL que desee utilizar copiando los algoritmos de cifrado y los programas de utilidad que necesite para trabajar con un certificado con autofirma:
   - Para el cifrado de 56 bits (utilizado con los programas bajo licencia 5722-CE2) copie /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip, cfwk.zip y ssltools.jar.
   - Para el cifrado de 128 bits (utilizado con los programas bajo licencia 5722-CE3) copie /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip, cfwk.zip y ssltools.jar.
3. Añada ssltools.jar y los archivos zip a la sentencia CLASSPATH.
4. Cree un directorio en el cliente denominado <SSL>\com\ibm\as400\access, donde <SSL> es el directorio donde ha copiado los archivos jar y zip.
5. Desde una solicitud de mandato dentro del directorio <SSL> en el cliente, ejecute el mandato siguiente:

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <nombresistema>:<puerto>
   ```

   donde <puerto> es el puerto de servidor de cualquiera de los servidores de sistema principal. Por ejemplo, puede utilizar 9476, que es el puerto por omisión del servidor de inicio de sesión segura en el iSeries.
6. »Escriba el número del certificado de autoridad certificadora (CA) que desea añadir al archivo de claves.« Asegúrese de que añade el certificado de CA y no el certificado del sitio.
7. Cuando se le pida que escriba un nombre de certificado, puede escribir cualquier serie alfanumérica.

   Nota: es necesario que ejecute KeyringDB para cada uno de los servidores que tenga un certificado con autofirma a fin de añadir cada uno de los certificados a la clase KeyRing. En cada iSeries que desee que utilice conexiones SSL, ejecute el mandato siguiente para añadir los certificados:

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <nombresistema>:<puerto>
   ```

Una vez llevadas a cabo las acciones anteriores, habrá terminado la puesta a punto de los certificados con autofirma. Ya puede ejecutar la aplicación, después de comprobar que la sentencia CLASSPATH contenga lo siguiente:

- el directorio que contiene com\ibm\as400\access\KeyRing.class
- jt400.jar
- sslightx.zip o sslightu.zip (según el archivo que haya bajado)

Como jt400.jar contiene la copia por omisión de KeyRing.class, el directorio que contiene com\ibm\as400\access\KeyRing.class debe estar en la sentencia CLASSPATH antes que jt400.jar.

Nota: en lugar de añadir el directorio que contiene el archivo KeyRing.class a la sentencia CLASSPATH, puede sustituir la clase antigua de jt400.jar por el nuevo archivo KeyRing.class.

Utilización de SSL para cifrar datos entre el cliente proxy y el servidor proxy

Puede emplear SSL para cifrar los datos que se intercambian entre el cliente proxy y el servidor proxy. Los archivos que se distribuyen con el programa bajo licencia IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) se utilizan para cifrar los datos. Del mismo modo que IBM Toolbox para Java, estos archivos son clases Java independientes de la plataforma que permiten al cliente proxy y al servidor proxy ejecutarse en cualquier plataforma con una máquina virtual Java.

Lleve a cabo las tareas siguientes para cifrar los datos que fluyen entre el cliente proxy y el servidor proxy:

1. Ponga a punto el servidor proxy para manejar los datos cifrados.
2. Ponga a punto el cliente proxy para manejar los datos cifrados.
3. Utilice el objeto SecureAS400 para forzar a IBM Toolbox para Java a cifrar datos.

Nota: los dos primeros pasos únicamente crean una vía de acceso segura entre el cliente proxy y el servidor proxy. La aplicación debe utilizar el objeto SecureAS400 para indicar a IBM Toolbox para Java que haga fluir los datos por la vía de acceso segura. De utilizar un objeto AS400 no se cifran los datos, sino que en su lugar se utiliza la vía de acceso normal hasta el servidor.

Si desea cifrar los datos que fluyen entre el cliente proxy y el servidor proxy, sólo necesita las clases Java que se distribuyen con el programa bajo licencia Client Encryption (5722-CE2 o 5722-CE3). Si desea cifrar los datos que fluyen entre el servidor proxy y los servidores iSeries, también debe poner a punto el cifrado entre el servidor proxy y el servidor iSeries.

Puesta a punto de SSL en el servidor proxy

Para habilitar SSL, el servidor proxy debe tener un certificado de servidor. Utilice la interfaz gráfica de usuario (GUI) IKeyman para crear el certificado de servidor que utilizará el servidor proxy. IKeyman es una herramienta de GUI, por lo que debe ejecutarla en una máquina cliente. Una vez que haya creado el certificado, puede copiarlo en el iSeries si el servidor proxy se ejecuta en el iSeries.

Para poner a punto el servidor proxy a fin de manejar datos cifrados, lleve a cabo las tareas siguientes:

1. Ponga a punto el cliente para ejecutar la GUI IKeyman.
2. Cree un certificado de servidor para el servidor proxy.
3. Inicie el servidor proxy con el certificado que acaba de crear.

Puesta a punto del cliente para ejecutar la GUI IKeyman

La GUI IKeyman es un programa Java basado en las interfaces Java Swing 1.1. Para utilizar IKeyman, el cliente debe ejecutar la máquina virtual Java de Java 1.1.8 (y el conector Swing 1.1) o la máquina virtual Java de Java 2.

La GUI IKeyman forma parte del programa bajo licencia IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) de ssltools.jar. El procedimiento que emplee para la puesta a punto del cliente a fin de utilizar SSL (y ejecutar IKeyman) depende de la versión del programa bajo licencia que ejecute.

Ponga a punto el cliente para utilizar SSL con los pasos siguientes:

1. Seleccione el directorio de la estación de trabajo donde desea colocar los archivos jar y zip necesarios.
2. Copie los archivos necesarios en el directorio seleccionado:
   - Si utiliza el cifrado de 56 bits, tras cargar el programa bajo licencia 5722-CE2 en el iSeries, copie los archivos siguientes en la estación de trabajo:
     - /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.sec
   - Si utiliza el cifrado de 128 bits, tras cargar el programa bajo licencia 5722-CE3 en el iSeries, copie los archivos siguientes en la estación de trabajo:
     - /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.sec
3. Añada el archivo jar y los archivos zip a la sentencia CLASSPATH. No añada el archivo .sec a la sentencia CLASSPATH.

   Nota: cfwk.zip debe ser el primer elemento de la sentencia CLASSPATH.

Creación de un certificado de servidor

Para crear un certificado con autofirma deberá utilizar la GUI IKeyman.

Nota: si se detiene la ejecución de la GUI IKeyman, compruebe que cfwk.zip sea el primer elemento de la sentencia CLASSPATH y que cfwk.sec esté en el mismo directorio que cfwk.zip.

Cree un certificado de servidor para el servidor proxy con los pasos siguientes:

1. Inicie la GUI IKeyman con el mandato siguiente:

```
java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
```

2. En el menú **Archivo de base de datos de claves** de IKeyman, seleccione **Nuevo**.
3. En el diálogo **Nuevo**, no modifique el valor de **Tipo de base de datos de claves**, que debe ser **Clase de base de datos de claves SSLight**.
4. Escriba un valor en **Nombre de archivo** (ProxyServerKeyring.class, por ejemplo) o pulse **Examinar** a fin de localizar el archivo de clase que desee utilizar para el archivo de claves.

   Nota: recuerde el nombre del archivo de claves, ya que lo necesitará para iniciar el servidor proxy seguro.
5. Escriba una vía de acceso en **Ubicación** o acepte la ubicación por omisión, que es el directorio de trabajo actual, y pulse **Aceptar**.
6. En el diálogo **Solicitud de contraseña**, escriba un valor en **Contraseña** y en **Confimar contraseña** y, a continuación, pulse **Aceptar**. (**Establecer hora de caducidad** es opcional y no es necesario seleccionarlo.)

   Nota: recuerde la contraseña, que necesitará para iniciar el servidor proxy seguro. Los iconos de claves de este diálogo representan la fiabilidad relativa de la contraseña. Para que una contraseña sea fiable es preciso combinar caracteres alfanuméricos en mayúsculas y en minúsculas.
7. En el menú de archivo **Crear** de IKeyman, seleccione **Nuevo certificado con autofirma**.
8. En el diálogo **Crear nuevo certificado con autofirma**, escriba una **Etiqueta de clave** (por ejemplo, MiCertificado) y una **Organización**.
9. Pulse la lista **País** para seleccionar un país o una región, escriba un **Periodo de validez** o acepte el valor por omisión; a continuación pulse **Aceptar**.
10. En el menú **Archivo de base de datos de claves** de IKeyman, seleccione **Cerrar** y, a continuación, en el mismo menú pulse **Salir**.

Ahora podrá ver el archivo de claves que acaba de crear en el directorio actual.

Inicio del servidor proxy con el nuevo certificado

Antes de iniciar el servidor proxy, compruebe que la sentencia CLASSPATH correspondiente al servidor proxy contiene jt400.jar, sslightx.zip y la ubicación del archivo de claves de servidor proxy.

Inicie el servidor proxy con el certificado que acaba de crear. Utilice los parámetros -keyringName y -keyringPassword para pasar esta información al servidor proxy. Por ejemplo:

```
java com.ibm.as400.access.ProxyServer -keyringName archclaveservproxy
-keyringPassword contrpxy
```

Puesta a punto de SSL en el cliente proxy

El procedimiento siguiente muestra los pasos que debe llevar a cabo para añadir el certificado de servidor a la base de datos de certificados en el cliente, que se almacena en un archivo .class de Java. Añadir el certificado de servidor al cliente es necesario ya que el servidor utiliza un certificado con autofirma.

Para poner a punto el cliente proxy a fin de intercambiar datos cifrados, realice las tareas siguientes:

1. Ponga a punto el servidor proxy para manejar los datos cifrados y, a continuación, inicie el servidor proxy.
2. Ponga a punto el cliente para utilizar SSL.
3. Utilice KeyringDB para obtener el certificado de servidor del servidor proxy.
4. Ponga a punto el cliente para utilizar el archivo KeyRing.class actualizado.
5. Establezca los valores de proxy seguro en el cliente.

Puesta a punto del cliente para utilizar SSL

La herramienta que baja el certificado (KeyringDB) es un programa Java. Para utilizar este programa, debe ejecutarse la máquina virtual Java (JVM) de Java 1.1.8 o Java 2 en el cliente. KeyringDB forma parte del programa bajo licencia IBM iSeries Client Encryption (5722-CE2 o 5722-CE3) de ssltools.jar. El procedimiento que emplee para la puesta a punto del cliente a fin de utilizar SSL depende de la versión del programa bajo licencia que ejecute.

Tras poner a punto el servidor proxy, ponga a punto el cliente para utilizar SSL con los pasos siguientes:

1. Seleccione el directorio de la estación de trabajo donde desea colocar los archivos jar y zip necesarios.
2. Copie los archivos necesarios en el directorio seleccionado:
   - Si utiliza el cifrado de 56 bits, tras cargar el programa bajo licencia 5722-CE2 en el iSeries, copie los archivos siguientes en la estación de trabajo:
     - /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip
   - Si utiliza el cifrado de 128 bits, tras cargar el programa bajo licencia 5722-CE3 en el servidor, copie los archivos siguientes en la estación de trabajo:
     - /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip
3. Añada el archivo jar y los archivos zip a la sentencia CLASSPATH.
4. Cree un directorio en el cliente denominado <SSL>\com\ibm\as400\access donde <SSL> es el directorio donde ha copiado los archivos jar y zip.

Adición del certificado de servidor para el servidor proxy

KeyringDB crea un nuevo archivo KeyRing.class que contiene el certificado de servidor y lo coloca en el subdirectorio com\ibm\as400\access del directorio actual.

Utilice la herramienta KeyringDB para añadir el certificado de servidor a KeyRing.class con los pasos siguientes:

1. Desde el directorio donde ha colocado los archivos jar y zip, ejecute el mandato siguiente:

```
java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect
nombreservidorproxy:puerto
```

donde:

- *nombreservidorproxy* es el nombre de la máquina que ejecuta el servidor proxy
- *puerto* es el puerto en el que escucha el servidor proxy seguro (por omisión el puerto 3471)

Por ejemplo:

```
java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect
myProxyServer:3471
```

2. Cuando se le pida qué certificado desea utilizar, elija el certificado del sitio 0.
3. Cuando se le pida que escriba un nombre de certificado, puede escribir cualquier serie alfanumérica.

Puesta a punto del cliente para utilizar el archivo KeyRing.class actualizado

El archivo jt400Proxy.jar contiene KeyRing.class. Para poner a punto el cliente a fin de utilizar el archivo KeyRing.class actualizado, compruebe que la sentencia CLASSPATH contenga lo siguiente:

- el directorio que contiene com\ibm\as400\access\KeyRing.class
- jt400Proxy.jar
- sslightx.zip o sslightu.zip (según el archivo que haya bajado)
- cfwk.zip

Como jt400Proxy.jar contiene la copia por omisión de KeyRing.class, el directorio que contiene com\ibm\as400\access\KeyRing.class debe estar en la sentencia CLASSPATH antes que jt400Proxy.jar.

Nota: en lugar de añadir el directorio que contiene el archivo KeyRing.class a la sentencia CLASSPATH, puede añadir el archivo KeyRing.class nuevo al archivo jt400Proxy.jar. Al añadir el archivo KeyRing.class nuevo a jt400Proxy.jar se sobrescribe la versión antigua.

Establecimiento de los valores de proxy seguro en el cliente

Para indicar al cliente proxy que se comunique con el servidor proxy por medio de una conexión segura, establezca las siguientes propiedades del sistema:

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

donde *servidorproxy* es el nombre de la máquina que ejecuta el servidor proxy

```
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=modalidad
```

donde *modalidad* es uno de los enteros siguientes:

- 1 para cifrar entre el cliente proxy y el servidor proxy
- 2 para cifrar entre el servidor proxy y el servidor iSeries
- 3 para cifrar entre el cliente proxy y el servidor proxy y entre el servidor proxy y el servidor iSeries

Por ejemplo, el mandato siguiente inicia una aplicación con SSL:

```
java -Dcom.ibm.as400.access.AS400.proxyServer=myProxyServer
   -Dcom.ibm.as400.access.SecureAS400.proxyEncryptionMode=1
myApplication
```

Servicios de autenticación

IBM Toolbox para Java proporciona clases que interactúan con los servicios de seguridad proporcionados por OS/400. Concretamente, se proporciona soporte para autenticar una identidad de usuario (al que a veces se hace referencia como *principal*) y una contraseña en relación con el registro de usuarios de OS/400. Entonces se puede establecer una credencial que represente al usuario autenticado. La credencial permite alterar la identidad de la hebra de OS/400 actual para que trabaje bajo las autorizaciones y los permisos del usuario autenticado. En efecto, este intercambio de identidad hace que la hebra actúe como si el usuario autenticado hubiese realizado un inicio de sesión.

Nota: los servicios para establecer e intercambiar credenciales sólo están soportados para los servidores cuyo release es igual o superior a V5R1M0.

Visión general del soporte proporcionado

El objeto AS400 proporciona autenticación para un determinado perfil de usuario y su contraseña en relación con el servidor. También se pueden recuperar para el sistema »tickets de kerberos y símbolos de perfil« que representen perfiles de usuario y contraseñas autenticados.

»**Nota:** para utilizar los tickets de kerberos debe instalarse J2SDK v1.4 y configurarse la interfaz de programación de aplicaciones Java General Security Services (JGSS). Para obtener más información sobre JGSS, consulte la documentación de seguridad de J2SDK v1.4 .«

»Para utilizar los tickets de kerberos, establezca únicamente el nombre de sistema (y no la contraseña) en el objeto AS400. La identidad de usuario se recupera mediante la infraestructura de JGSS. Sólo puede establecer un método de autenticación en un objeto AS400 a la vez. Al establecer la contraseña se borra el ticket de kerberos o el símbolo de perfil.«

Para utilizar los símbolos de perfil, utilice los métodos getProfileToken() para recuperar instancias de la clase ProfileTokenCredential. El símbolo de perfil puede describirse como la representación de un perfil de usuario y una contraseña autenticados para un servidor específico. Los símbolos de perfil caducan con el tiempo (duran, como máximo, una hora), pero en algunos casos se pueden renovar para que su tiempo de vida sea más largo.

»El ejemplo siguiente crea un objeto del sistema y utiliza ese objeto para generar un símbolo de perfil. A continuación, el ejemplo utiliza el símbolo de perfil para crear otro objeto del sistema y emplea el segundo objeto del sistema para conectar con el servicio de mandatos:

```
AS400 system = new AS400("mySystemName", "MYUSERID", "MYPASSWORD");
ProfileTokenCredential myPT = system.getProfileToken();
AS400 system2 = new AS400("mySystemName", myPT);
system2.connectService(AS400.COMMAND); «
```

Establecer identidades de hebra

Una credencial se puede establecer ya sea en un contexto remoto o en un contexto local. Una vez creada, se puede serializar o distribuir según lo requiera la aplicación llamadora. Se puede utilizar una credencial, cuando se pasa a un proceso en ejecución en el servidor asociado, para modificar o intercambiar (*swap*) la identidad de la hebra de OS/400 y llevar a cabo tareas en nombre del usuario previamente autenticado.

Una aplicación práctica de este soporte sería en una aplicación de dos niveles: en el primer nivel (por ejemplo, un PC), una interfaz gráfica de usuario realizaría la autenticación de un perfil de usuario y una contraseña, y en el segundo nivel (el servidor) se llevaría a cabo el trabajo de ese usuario. Al utilizar credenciales de símbolo de perfil (clases ProfileTokenCredential), la aplicación puede evitar el tener que pasar directamente los ID de usuario y las contraseñas a través de la red. Entonces, el símbolo de perfil se puede distribuir al programa del segundo nivel, que puede realizar el intercambio (método *swap()*) y operar bajo las autorizaciones y los permisos de OS/400 asignados al usuario.

Nota: los símbolos de perfil, debido a su tiempo de vida limitado, son inherentemente más seguros que pasar un perfil de usuario y una contraseña; sin embargo, la aplicación aún los debe considerar como información delicada y ha de manejarlos como tal. Debido a que el símbolo representa a un usuario autenticado y su contraseña, una aplicación hostil podría aprovecharse de él para trabajar en nombre de ese usuario. La aplicación es, en última instancia, la que debe encargarse de que el acceso a las credenciales se realice de manera segura.

Ejemplo

En este código encontrará un ejemplo de cómo se utiliza una credencial de símbolo de perfil para intercambiar la identidad de la hebra de OS/400 y llevar a cabo tareas en nombre de un determinado usuario.

Clases de servlets

Las clases de servlets que se proporcionan con IBM Toolbox para Java trabajan con las clases de acceso (que se encuentran en el servidor Web) para dar acceso a la información ubicada en el servidor iSeries. Usted es quien decide de qué manera va a utilizar las clases de servlets como ayuda para sus propios proyectos de servlets.

En el siguiente diagrama se ve cómo las clases de servlets funcionan entre el navegador, el servidor Web y los datos de iSeries. Un navegador se conecta al servidor Web que está ejecutando el servlet. Los »archivos jt400Servlet.jar y jt400.jar« residen en el servidor Web, porque las clases de servlets utilizan algunas de las clases de acceso para recuperar los datos y las clases HTML para presentar los datos. El servidor Web se conecta al sistema iSeries en el que están los datos.

Figura 1: cómo funcionan los servlets

Hay cuatro tipos de clases de servlets en IBM Toolbox para Java:

- Clases de autenticación
- Clases RowData
- Clases RowMetaData
- Clases Converter

Nota: el archivo jt400Servlet.jar incluye tanto las clases HTML como las clases de servlets. »Debe actualizar la CLASSPATH para que señale a los dos archivos (jt400Servlet.jar y jt400.jar) si desea utilizar las clases de los paquetes com.ibm.as400.util.html y com.ibm.as400.util.servlet.«

Encontrará más información sobre los servlets en general en la sección Consulta o referencia.

Clases de autenticación

Dos clases del paquete de servlets llevan a cabo la autenticación de servlets: AuthenticationServlet y AS400Servlet.

Clase AuthenticationServlet

AuthenticationServlet es una implementación de HttpServlet que lleva a cabo la autenticación básica de servlets.Las subclases de AuthenticationServlet alterarán temporalmente uno o varios de los métodos siguientes:

- Altere temporalmente el método validateAuthority() para realizar la autenticación (obligatorio)
- Altere temporalmente el método bypassAuthentication() para que la subclase autentique únicamente determinadas peticiones
- Altere temporalmente el método postValidation() para permitir el proceso adicional de la petición tras la autenticación

La clase AuthenticationServlet proporciona métodos que permiten llevar a cabo estas acciones:

- Inicializar el servlet
- Obtener el ID de usuario autenticado
- Establecer un ID de usuario tras eludir la autenticación
- Anotar excepciones y mensajes

Clase AS400Servlet

La clase AS400Servlet es una subclase abstracta de AuthenticationServlet que representa un servlet HTML.Puede utilizar una agrupación de conexiones para compartir conexiones y gestionar el número de conexiones que un usuario de servlet puede tener con el servidor.

La clase AS400Servlet proporciona métodos que permiten llevar a cabo estas acciones:

- Validar la autoridad de usuario (alterando temporalmente el método validateAuthority() de la clase AuthenticationServlet)
- Conectarse a un sistema
- Obtener objetos de agrupación de conexiones de la agrupación y devolver objetos de agrupación de conexiones a la agrupación
- Cerrar una agrupación de conexiones
- Obtener y establecer los códigos head de documentos HTML
- Obtener y establecer los códigos end de documentos HTML

Encontrará más información sobre los servlets en general en la sección Consulta o referencia.

Clase RowData

La clase RowData es una clase abstracta que proporciona una manera de describir una lista de datos y acceder a ella.

Estas son las cuatro clases principales que amplían la clase RowData:

- ListRowData
- RecordListRowData
- »ResourceListRowData«
- SQLResultSetRowData

Las clases RowData le permiten realizar estas tareas:

- Obtener y establecer la posición actual
- Obtener los datos de fila situados en una columna dada utilizando el método getObject()
- Obtener los metadatos de la fila
- Obtener o establecer las propiedades de un objeto situado en una columna dada
- Obtener el número de filas de la lista mediante el método length()

Posición de RowData

Hay varios métodos que permiten obtener y establecer la posición actual dentro de una lista. la tabla siguiente indica los métodos para establecer y obtener correspondientes a las clases RowData.

Métodos para establecer		**Métodos para obtener**
absolute()	next()	getCurrentPosition()
afterLast()	previous()	isAfterLast()
beforeFirst()	relative()	isBeforeFirst()
first()		isFirst()
last()		isLast()

Clase ListRowData

La clase ListRowData le permite llevar a cabo estas tareas:

- Añadir filas a la lista de resultados y eliminar filas de la lista de resultados.
- Obtener y establecer la fila.
- Obtener información sobre las columnas de la lista con el método getMetaData().
- Establecer información de columna con el método setMetaData().

La clase ListRowData representa una lista de datos. Mediante las clases de acceso de IBM Toolbox para Java, la clase ListRowData puede representar numerosos tipos de información, entre ellos los siguientes:

- Un directorio del sistema de archivos integrado
- Una lista de trabajos
- Una lista de mensajes de una cola de mensajes
- Una lista de usuarios
- Una lista de impresoras
- Una lista de archivos en spool

Este ejemplo muestra cómo funcionan las clases ListRowData y HTMLTableConverter. Muestra el código Java, el código HTML y el aspecto HTML.

Clase RecordListRowData

La clase RecordListRowData le permite realizar estas tareas:

- Añadir filas a la lista de registros y eliminar filas de la lista de registros.
- Obtener y establecer la fila.
- Establecer el formato de registro con el método setRecordFormat.
- Obtener el formato de registro.

La clase RecordListRowData representa una lista de registros. Los registros se pueden obtener del servidor con distintos formatos, tales como:

- Un registro que deba leerse de un archivo del servidor o escribirse en él
- Una entrada de una cola de datos
- Los datos de parámetro de una llamada a programa
- Los datos que se devuelvan y necesiten convertirse entre el formato del servidor y el formato Java

Este ejemplo muestra cómo funcionan las clases RecordListRowData y HTMLTableConverter. Muestra el código Java, el código HTML y el aspecto HTML.

Clase ResourceListRowData

La clase ResourceListRowData representa una lista de recursos de datos. Utilice objetos ResourceListRowData para representar cualquier implementación de la interfaz ResourceList.

A las listas de recursos se les da el formato de una serie de filas y cada una de las filas contiene un número finito de columnas que viene determinado por la cantidad de ID de atributo de columna. Cada columna de una fila contiene un elemento de datos individual.

La clase ResourceListRowData proporciona métodos que permiten llevar a cabo las acciones siguientes:

- Obtener y establecer los ID de atributo de columna
- Obtener y establecer la lista de recursos
- Recuperar el número de filas de la lista
- Obtener los datos de columna de la fila actual
- Obtener la lista de propiedades del objeto de datos
- Obtener los metadatos de la lista

Ejemplo: presentar una lista de recursos en un servlet

Clase QLResultSetRowData

La clase SQLResultSetRowData representa un conjunto de resultados de SQL en forma de lista de datos. Estos datos los genera una sentencia SQL mediante JDBC. Con los métodos proporcionados, puede obtener y establecer los metadatos del conjunto de resultados.

Este ejemplo muestra cómo funcionan las clases ListRowData y HTMLTableConverter. Muestra el código Java, el código HTML y el aspecto HTML.

Clases RowMetaData

La clase RowMetaData define una interfaz que se utiliza para obtener información acerca de las columnas de un objeto RowData.

Con las clases RowMetaData puede hacer estas tareas:

- Obtener el número de columnas
- Obtener el nombre, tipo o tamaño de la columna
- Obtener o establecer la etiqueta de columna
- Obtener la precisión o escala de los datos de la columna
- Determinar si los datos de una columna son datos de texto

La clase RowMetaData está implementada por tres clases principales. Estas clases proporcionan todas las funciones de RowMetaData listadas más arriba y tienen, además, sus propias funciones específicas:

- ListMetaData
- RecordFormatMetaData
- SQLResultSetMetaData

Clase ListMetaData

La clase ListMetaData permite obtener información acerca de las columnas de una clase ListRowData y cambiar sus valores. Utiliza el método setColumns() para establecer el número de columnas, borrando la información de columna que pudiera haber antes. Alternativamente, también puede usted pasar el número de columnas cuando establezca los parámetros del constructor.

Este ejemplo muestra cómo funcionan las clases ListMetaData, ListRowData y HTMLTableConverter. Muestra el código Java, el código HTML y el aspecto HTML.

Clase RecordFormatMetaData

La clase RecordFormatMetaData utiliza la clase RecordFormat de IBM Toolbox para Java. Le permite proporcionar el formato de registro cuando establezca los parámetros del constructor o bien utilizar los métodos get y set para acceder al formato de registro.

El ejemplo siguiente muestra cómo se crea un objeto RecordFormatMetaData:

```
// Cree un objeto RecordFormatMetaData a partir de un formato de registro de
un archivo secuencial.
RecordFormat recordFormat = sequentialFile.getRecordFormat();
RecordFormatMetaData metadata = new RecordFormatMetaData(recordFormat);

// Visualice los nombres de las columnas del archivo.
int numberOfColumns = metadata.getColumnCount();
for (int column=0; column < numberOfColumns; column++)
{
   System.out.println(metadata.getColumnName(column));
}
```

Clase SQLResultSetMetaData

La clase SQLResultSetMetaData devuelve información acerca de las columnas de un objeto SQLResultSetRowData. Puede proporcionar el conjunto de resultados cuando establezca los parámetros del constructor o bien utilizar los métodos get y set para acceder a los metadatos del conjunto de resultados.

El ejemplo de código siguiente muestra cómo se crea un objeto SQLResultSetMetaData:

```
// Cree un objeto SQLResultSetMetaData a partir de los metadatos del
conjunto de resultados.
SQLResultSetRowData rowdata = new SQLResultSetRowData(resultSet);
SQLResultSetMetaData sqlMetadata  = rowdata.getMetaData();

// Visualice la precisión de las columnas que no sean de texto.
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("La columna: " + name + " contiene datos de
texto.");
   }
         else
   {
      System.out.println("La columna: " + name + " tiene una precisión igual
a " + sqlMetadata.getPrecision(column));
   }
}
```

Clases conversoras

Las clases conversoras permiten convertir los datos de las filas en matrices de tipo serie con formato. El resultado tiene formato HTML y está preparado para presentarse en una página HTML. Las clases que se encargan de convertir son las siguientes:

- StringConverter
- HTMLFormConverter
- HTMLTableConverter

Clase StringConverter

La clase StringConverter es una clase abstracta que representa un conversor de tipo serie de los datos de una fila. Proporciona un método convert() para convertir los datos de la fila. Este método devuelve una representación en forma de matriz de series de los datos de esa fila.

Clase HTMLFormConverter

La clase HTMLFormConverter amplía la clase StringConverter proporcionando un método de conversión adicional llamado convertToForms(). Este método convierte los datos de las filas en una matriz de tablas HTML de una sola fila. Estos códigos de tabla le permiten visualizar la información formateada en un navegador.

Para adaptar el aspecto del formulario HTML, puede utilizar los diversos métodos get y set a fin de ver o cambiar los atributos del formulario. Por ejemplo, entre los atributos que puede establecer se encuentran los siguientes:

- Alineación
- Espaciado de casilla
- Hiperenlaces de cabecera
- Anchura

Ejemplos

Ejemplo: cómo se utiliza HTMLFormConverter. (Compile y ejecute este ejemplo con un servidor Web en ejecución.)

Clase HTMLTableConverter

La clase HTMLTableConverter amplía la clase StringConverter al proporcionar un método convertToTables(). Este método convierte los datos de las filas en una matriz de tablas HTML que un servlet puede utilizar para visualizar la lista en un navegador.

Los métodos getTable() y setTable() le permiten elegir una tabla por omisión que se utilizará durante la conversión. Puede establecer cabeceras de tabla dentro del objeto tabla HTML o puede utilizar los metadatos para la información de cabecera, estableciendo para ello el valor true en el método setUseMetaData().

El método setMaximumTableSize() le permite limitar el número de filas de una sola tabla. Si todos los datos de las filas no caben en el tamaño de tabla especificado, el conversor generará otro objeto tabla HTML en la matriz de salida. Este proceso continuará hasta que se hayan convertido los datos de todas las filas.

Ejemplos

Los ejemplos que hay a continuación muestran cómo se utiliza la clase HTMLTableConverter:

- Ejemplo: cómo se utiliza ListRowData
- Ejemplo: cómo se utiliza RecordListRowData
- Ejemplo: cómo se utiliza SQLResultSetRowData
- Ejemplo: presentar un objeto ResourceList en un servlet

Clases de utilidades

Las clases de utilidades ayudan a llevar a cabo tareas específicas. IBM Toolbox para Java ofrece las siguientes utilidades:

- AS400ToolboxInstaller: permite instalar y actualizar las clases de IBM Toolbox para Java en el cliente. Esta función está disponible como programa Java y también tiene una API (interfaz de programas de aplicación).
- AS400ToolboxJarMaker: genera un archivo JAR de IBM Toolbox para Java que se carga más deprisa; para ello, crea un archivo JAR más pequeño a partir de uno más grande o descomprime selectivamente un archivo JAR para obtener acceso a los archivos de contenido individuales.
- CommandPrompter: solicita el parámetro en un mandato determinado. CommandPrompter ofrece funciones parecidas a la solicitud de mandatos CL de iSeries (pulsando F4) y las mismas funciones que la solicitud de mandatos de Management Central.
- RunJavaApplication y VRunJavaApplication: permiten ejecutar un programa Java en un servidor iSeries desde una solicitud de línea de mandatos.
- JPing: permite consultar un servidor para averiguar qué servicios están activos. También puede especificar si desea realizar una operación ping de los puertos SSL.

Clases de instalación y actualización en cliente

Se puede hacer referencia a las clases de IBM Toolbox para Java en la ubicación que tienen en el sistema de archivos integrado en el servidor. Dado que los arreglos temporales de programa (PTF) se aplican a esta ubicación, los programas Java que acceden directamente a estas clases en el servidor reciben estas actualizaciones de modo automático. El acceso a las clases desde el servidor no siempre funciona, en concreto en las situaciones siguientes:

- Si el enlace de comunicaciones que conecta el servidor y el cliente es de baja velocidad, el rendimiento que supone cargar las clases desde el servidor puede ser inaceptable.
- Si las aplicaciones Java utilizan la variable de entorno CLASSPATH para acceder a las clases que hay en el sistema de archivos del cliente, es necesario que iSeries Access para Windows redirija al servidor las llamadas al sistema de archivos. Tal vez no sea posible que iSeries Access para Windows resida en el cliente.

En estos casos, instalar las clases en el cliente es una solución mejor. La clase AS400ToolboxInstaller proporciona las funciones de instalación y actualización en cliente para gestionar las clases de IBM Toolbox para Java cuando residen en un cliente.

Utilización de AS400ToolboxInstaller

»El objeto AS400ToolboxInstaller es tanto un programa como una interfaz programable. El objeto tiene un método main() para que pueda ejecutarse desde la línea de mandatos. También tiene métodos de trabajador público a fin de que pueda incluirse en la aplicación y llamarse desde ella.

El objeto AS400ToolboxInstaller puede emplearse para instalar los archivos de la Caja de Herramientas en el cliente y mantenerlos actualizados. Cuando se utilizan por primera vez los archivos de la Caja de Herramientas, se copian del servidor en la estación de trabajo. Cuando se aplican arreglos PTF al servidor, el objeto AS400ToolboxInstaller actualiza los archivos de la estación de trabajo con los archivos nuevos del servidor.«

La clase AS400ToolboxInstaller copia archivos en el sistema de archivos local del cliente. Esta clase puede no funcionar en un applet; hay muchos navegadores que no permiten a un programa Java escribir en el sistema de archivos local.

Ejecutar la clase AS400ToolboxInstaller desde la línea de mandatos

La clase AS400ToolboxInstaller se puede utilizar como programa autónomo que se ejecuta desde la línea de mandatos. Que la clase AS400ToolboxInstaller se ejecute desde la línea de mandatos quiere decir que no tiene usted que escribir un programa. En vez de ello, ejecute la clase como aplicación Java para instalar, desinstalar o actualizar las clases de IBM Toolbox para Java.

Especificando según convenga la opción de instalar, desinstalar o comparar, invoque la clase AS400ToolboxInstaller con el siguiente mandato:

```
java utilities.AS400ToolboxInstaller [opciones]
```

La opción **-source** indica dónde se pueden encontrar las clases de IBM Toolbox para Java y la opción **-target** indica dónde se han de almacenar en el cliente las clases de IBM Toolbox para Java.

También hay opciones para instalar toda la caja de herramientas (Toolbox) o solo ciertas funciones. »Por ejemplo, para instalar o actualizar las clases del paquete de acceso de Toolbox para Java (jt400.jar) en la estación de trabajo, utilice el mandato siguiente:

```
java utilities.AS400ToolboxInstaller -install -package ACCESS -source
myAS400    -target c:\toolbox
```

En el ejemplo se supone que c:\toolbox es el directorio que contiene los archivos jar de Toolbox para Java.«

Incorporar la clase AS400ToolboxInstaller en el programa

La clase AS400ToolboxInstaller proporciona las interfaces de programas de aplicación (API) necesarias para instalar, desinstalar y actualizar en el cliente las clases de IBM Toolbox para Java desde dentro del programa.

Utilice el método install() para instalar o actualizar las clases de IBM Toolbox para Java. Para instalar o actualizar, proporcione las vías de acceso origen y destino, así como el nombre del paquete de clases en el programa Java. El URL origen señala a la ubicación de los archivos de control, en el servidor. La estructura de directorios se copia en el cliente desde el servidor.

El método install() solo copia archivos; **no** actualiza la variable de entorno CLASSPATH. Si el método install() se ejecuta satisfactoriamente, el programa Java puede efectuar una llamada al método getClasspathAdditions() para determinar lo que debe añadirse a la variable de entorno CLASSPATH.

El ejemplo que hay a continuación muestra cómo se utiliza la clase AS400ToolboxInstaller para instalar archivos desde un servidor llamado "mySystem" en el directorio "jt400" de la unidad d: y, a continuación, cómo se determina qué es lo que debe añadirse a la variable de entorno CLASSPATH:

```
         // Instale las clases de IBM Toolbox para Java
         // en el cliente.
     URL sourceURL = new
URL("http://mySystem.myCompany.com/QIBM/ProdData/HTTP/Public/jt400/");

     if (AS400ToolboxInstaller.install(
              "ACCESS",
              "d:\\jt400",
              sourceURL))

     {
         // Si las clases de IBM Toolbox para Java se han instalado
         // o actualizado, averigüe lo que debe añadirse a la
         // variable de entorno CLASSPATH.
        Vector additions = AS400ToolboxInstaller.getClasspathAdditions();

         // Si deben realizarse actualizaciones en CLASSPATH.
        if (additions.size() > 0)
        {
         // ...Procese cada adición de classpath.
        }
     }

         // ...En caso contrario, no se necesitaron actualizaciones.
```

El método isInstalled() permite determinar si las clases de IBM Toolbox para Java ya están instaladas en el cliente. El método isInstalled() le permite determinar si desea completar ahora la instalación o si prefiere aplazarla para otro momento.

El método install() instala y actualiza archivos en el cliente. Un programa Java puede efectuar una llamada al método isUpdateNeeded() para determinar si es necesaria una actualización antes de realizar una llamada a install().

Utilice el método unInstall() para eliminar del cliente las clases de IBM Toolbox para Java. El método unInstall solo elimina archivos; no realiza ningún cambio en la variable de entorno CLASSPATH. Haga una llamada al método getClasspathRemovals() para determinar lo que puede eliminarse de la variable de entorno CLASSPATH.

Si desea obtener más ejemplos de cómo se instala y actualiza la clase AS400ToolboxInstaller dentro de un programa en la estación de trabajo cliente, consulte el ejemplo de instalación/actualización.

AS400ToolboxJarMaker

Mientras que el formato de archivo JAR se diseñó para agilizar la bajada de los archivos de programa Java, la clase AS400ToolboxJarMaker genera una carga aún más rápida de un archivo JAR de IBM Toolbox para Java mediante su capacidad de crear un archivo JAR más pequeño a partir de uno más grande.

Además, la clase AS400ToolboxJarMaker puede descomprimir un archivo JAR para que así sea posible acceder a los archivos de contenido individual para uso básico.

Flexibilidad de AS400ToolboxJarMaker

Todas las funciones de AS400ToolboxJarMaker se llevan a cabo con la clase JarMaker y la subclase AS400ToolboxJarMaker:

- La herramienta JarMaker genérica funciona en cualquier archivo JAR o Zip; subdivide un archivo jar o reduce el tamaño de un archivo jar eliminando las clases que no se usan.
- La clase AS400ToolboxJarMaker personaliza y amplía las funciones de JarMaker para facilitar su uso con los archivos JAR de IBM Toolbox para Java.

En función de sus necesidades, puede invocar los métodos de AS400ToolboxJarMaker desde dentro del programa Java o desde una línea de mandatos. Para efectuar una llamada a AS400ToolboxJarMaker desde la línea de mandatos, utilice la sintaxis que se indica a continuación:

```
java utilities.JarMaker [opciones]
```

donde

- opciones = una o varias de las opciones disponibles

Si desea obtener un conjunto completo de las opciones disponibles para ejecutarse en una solicitud de línea de mandatos, consulte:

- Opciones para la clase base JarMaker
- Opciones ampliadas para la subclase AS400ToolboxJarMaker

Utilización de AS400ToolboxJarMaker

Descompresión de un archivo JAR

Suponga que desea descomprimir un único archivo empaquetado dentro de un archivo JAR. AS400ToolboxJarMaker le permite expandir el archivo en uno de estos directorios:

- Directorio actual (extract(jarFile))
- Otro directorio (extract(jarFile, outputDirectory))

Por ejemplo, el código siguiente hace que del archivo jt400.jar se extraigan AS400.class y todas sus clases dependientes:

```
java utilities.AS400ToolboxJarMaker -source jt400.jar
    -extract outputDir
    -requiredFile com/ibm/as400/access/AS400.class
```

Subdivisión de un archivo JAR individual en varios archivos JAR más pequeños

Suponga que desea subdividir un archivo JAR de gran tamaño en archivos JAR más pequeños, en función de su preferencia para el tamaño máximo de archivo JAR. AS400ToolboxJarMaker, de acuerdo con ello, le proporciona la función split(jarFile, splitSize).

En el código que figura a continuación, jt400.jar se subdivide en una serie de archivos JAR más pequeños, ninguno de los cuales tiene más de 300 K:

```
java utilities.AS400ToolboxJarMaker -split 300
```

Eliminación de archivos no utilizados de un archivo JAR Con AS400ToolboxJarMaker, puede excluir todos los archivos de IBM Toolbox para Java que su aplicación no necesite; para ello, basta con que seleccione únicamente los componentes, idiomas y CCSID de IBM Toolbox para Java necesarios para que se ejecute la aplicación. AS400ToolboxJarMaker también le proporciona la opción de incluir o excluir los archivos de JavaBean asociados a los componentes que ha elegido.

Por ejemplo, el mandato siguiente crea un archivo JAR que contiene únicamente las clases de IBM Toolbox para Java necesarias para que funcionen los componentes CommandCall y ProgramCall de IBM Toolbox para Java:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall
```

Además, si no es necesario hacer que el texto se convierta entre Unicode y las tablas de conversión del juego de caracteres de doble byte (DBCS), se puede crear un archivo JAR cuyo tamaño tenga 400 Kbytes menos tan solo con omitir las tablas de conversión innecesarias con la opción -ccsid:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall
-ccsid 61952
```

Nota: las clases conversoras no se incluyen junto con las clases de llamada a programa. Al incluir las clases de llamada a programa, también deberá incluir explícitamente las clases conversoras utilizadas por el programa por medio de la opción -ccsid.

Componentes soportados por IBM Toolbox para Java

A continuación figuran los ID de componente que puede especificar al invocar la herramienta AS400ToolboxJarMaker.

- La primera columna muestra el nombre común del componente.
- La segunda columna indica la palabra clave que debe especificar al utilizar el código de opción `-component`.
- La tercera columna muestra el valor de Integer que debe especificar en setComponents() y getComponents().

Componente	Palabra clave	Constante
Objeto de servidor	AS400	AS400ToolboxJarMaker.AS400
Llamada a mandato	CommandCall	AS400ToolboxJarMaker.COMMAND_CALL
Agrupación de conexiones	ConnectionPool	AS400ToolboxJarMaker.CONNECTION_POOL
Áreas de datos	DataArea	AS400ToolboxJarMaker.DATA_AREA
Conversión y descripción de datos	DataDescription	AS400ToolboxJarMaker.DATA_DESCRIPTION
Colas de datos	DataQueue	AS400ToolboxJarMaker.DATA_QUEUE
Certificados digitales	DigitalCertificate	AS400ToolboxJarMaker.DIGITAL_CERTIFICATE
FTP	FTP	AS400ToolboxJarMaker.FTP
Sistema de archivos integrado	IntegratedFileSystem	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM
JAAS	JAAS	AS400ToolboxJarMaker.JAAS
Llamada a aplicación Java	JavaApplicationCall	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL
JDBC	JDBC	AS400ToolboxJarMaker.JDBC
Trabajos y colas de trabajos	Job	AS400ToolboxJarMaker.JOB
Mensajes y colas de mensajes	Message	AS400ToolboxJarMaker.MESSAGE
Tipos de datos numéricos	NumericDataTypes	AS400ToolboxJarMaker.NUMERIC_DATA_TYPES
NetServer	NetServer	AS400ToolboxJarMaker.NETSERVER
Impresión de red	Imprimir	AS400ToolboxJarMaker.PRINT
Llamada a programa	ProgramCall	AS400ToolboxJarMaker.PROGRAM_CALL
Acceso a nivel de registro	RecordLevelAccess	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS
Servidor seguro	SecureAS400	AS400ToolboxJarMaker.SECURE_AS400
Llamada a programa de servicio	ServiceProgramCall	AS400ToolboxJarMaker.SERVICE_PROGRAM_CALL

Estado del sistema	SystemStatus	AS400ToolboxJarMaker.SYSTEM_STATUS
Valores del sistema	SystemValue	AS400ToolboxJarMaker.SYSTEM_VALUE
Rastreo y anotaciones	Trace	AS400ToolboxJarMaker.TRACE
Usuarios y grupos	por omisión	AS400ToolboxJarMaker.USER
Espacios de usuario	UserSpace	AS400ToolboxJarMaker.USER_SPACE
Objeto de servidor visual	AS400Visual	AS400ToolboxJarMaker.AS400_VISUAL
Llamada a mandato visual	CommandCallVisual	AS400ToolboxJarMaker.COMMAND_CALL_VISUAL
Colas de datos visuales	DataQueueVisual	AS400ToolboxJarMaker.DATA_QUEUE_VISUAL
Sistema de archivos integrado visual	IntegratedFileSystemVisual	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM_VISUAL
Llamada a aplicación Java visual	JavaApplicationCallVisual	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL_VISUAL
JDBC visual	JDBCVisual	AS400ToolboxJarMaker.JDBC_VISUAL
Trabajos y colas de trabajos visuales	JobVisual	AS400ToolboxJarMaker.JOB_VISUAL
Mensajes y colas de mensajes visuales	MessageVisual	AS400ToolboxJarMaker.MESSAGE_VISUAL
Impresión de red visual	PrintVisual	AS400ToolboxJarMaker.PRINT_VISUAL
Llamada a programa visual	ProgramCallVisual	AS400ToolboxJarMaker.PROGRAM_CALL_VISUAL
Acceso a nivel de registro visual	RecordLevelAccessVisual	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS_VISUAL
Usuarios y grupos visuales	UserVisual	AS400ToolboxJarMaker.USER_VISUAL

Valores de CCSID y codificación soportados por IBM Toolbox para Java

IBM Toolbox para Java se distribuye con un conjunto de tablas de conversión, denominadas según el CCSID. Las clases de IBM Toolbox para Java (como por ejemplo CharConverter) utilizan internamente estas tablas al convertir los datos transferidos a un servidor iSeries o AS/400e o desde éste. Por ejemplo, la tabla de conversión correspondiente al CCSID 1027 está en el archivo `com/ibm/as400/access/ConvTable1027.class`. Las tablas de conversión de los CCSID siguientes se incluyen en el archivo jar de IBM Toolbox para Java; con JDK se da soporte a otras codificaciones. El servidor central del servidor ya no se utiliza para bajar tablas en tiempo de ejecución. Cualquier CCSID especificado para el que no se encuentre una tabla de conversión o una codificación de JDK hará que se genere una excepción. Algunas de estas tablas pueden ser redundantes respecto a las tablas incluidas en JDK. En la actualidad IBM Toolbox para Java da soporte a 122 CCSID de iSeries y AS/400e distintos.

Encontrará más información acerca de los CCSID, así como una lista completa de los CCSID reconocidos por los servidores iSeries y AS/400e, en el tema sobre globalización del Information Center.

CCSID soportados en IBM Toolbox para Java

CCSID	Formato	Descripción
37	EBCDIC de un solo byte	Estados Unidos y otros
273	EBCDIC de un solo byte	Austria, Alemania
277	EBCDIC de un solo byte	Dinamarca, Noruega
278	EBCDIC de un solo byte	Finlandia, Suecia
280	EBCDIC de un solo byte	Italia
284	EBCDIC de un solo byte	España, Latinoamérica
285	EBCDIC de un solo byte	Reino Unido
290	EBCDIC de un solo byte	Japonés Katakana (únicamente de un solo byte)
297	EBCDIC de un solo byte	Francia
300	EBCDIC de doble byte	Japonés gráfico (subconjunto de 16684)
367	ASCII/ISO/Windows	ASCII (estándar ANSI X3.4)
420	EBCDIC de un solo byte (bidireccional)	Árabe EBCDIC ST4
423	EBCDIC de un solo byte	Griego (para compatibilidad, véase 875)
424	EBCDIC de un solo byte (bidireccional)	Hebreo EBCDIC ST4
437	ASCII/ISO/Windows	ASCII (Datos de PC EE.UU.)
500	EBCDIC de un solo byte	Latino 1 (MNCS)
720	ASCII/ISO/Windows	Árabe (MS-DOS)
737	ASCII/ISO/Windows	Griego (MS-DOS)
775	ASCII/ISO/Windows	Báltico (MS-DOS)
813	ASCII/ISO/Windows	ISO 8859-7 (Grecolatino)
819	ASCII/ISO/Windows	ISO 8859-1 (Latino 1)
833	EBCDIC de un solo byte	Coreano (únicamente de un solo byte)
834	EBCDIC de doble byte	Coreano gráfico (subconjunto de 4930)
835	EBCDIC de doble byte	Chino tradicional gráfico
836	EBCDIC de un solo byte	Chino simplificado (únicamente de un solo byte)
837	EBCDIC de doble byte	Chino simplificado gráfico

838	EBCDIC de un solo byte	Tailandés
850	ASCII/ISO/Windows	Latino 1
851	ASCII/ISO/Windows	Griego
852	ASCII/ISO/Windows	Latino 2
855	ASCII/ISO/Windows	Cirílico
857	ASCII/ISO/Windows	Turco
860	ASCII/ISO/Windows	Portugués
861	ASCII/ISO/Windows	Islandés
862	ASCII/ISO/Windows (bidireccional)	Hebreo ASCII ST4
863	ASCII/ISO/Windows	Canadá
864	ASCII/ISO/Windows (bidireccional)	Árabe ASCII ST5
865	ASCII/ISO/Windows	Dinamarca/Noruega
866	ASCII/ISO/Windows	Cirílico/Ruso
869	ASCII/ISO/Windows	Griego
870	EBCDIC de un solo byte	Latino 2
871	EBCDIC de un solo byte	Islandés
874	ASCII/ISO/Windows	Tailandés (subconjunto de 9066)
875	EBCDIC de un solo byte	Griego
878	ASCII/ISO/Windows	Ruso
880	EBCDIC de un solo byte	Cirílico multilingüe (para compatibilidad, véase 1025)
912	ASCII/ISO/Windows	ISO 8859-2 (Latino 2)
914	ASCII/ISO/Windows	ISO 8859-4 (Latino 4)
915	ASCII/ISO/Windows	ISO 8859-5 (cirílico de 8 bits)
916	ASCII/ISO/Windows (bidireccional)	ISO 8859-8 (hebreo) ST5
920	ASCII/ISO/Windows	ISO 8859-9 (Latino 5)
921	ASCII/ISO/Windows	ISO 8859-13 (báltico de 8 bits)
922	ASCII/ISO/Windows	Estonia ISO-8
923	ASCII/ISO/Windows	ISO 8859-15 (Latino 9)
930	EBCDIC de byte mixto	Japonés (subconjunto de 5026)
933	EBCDIC de byte mixto	Coreano (subconjunto de 1364)
935	EBCDIC de byte mixto	Chino simplificado (subconjunto de 1388)
937	EBCDIC de byte mixto	Chino tradicional
939	EBCDIC de byte mixto	Japonés (subconjunto de 5035)
1025	EBCDIC de un solo byte	Cirílico
1026	EBCDIC de un solo byte	Turco
1027	EBCDIC de un solo byte	Japonés latino (únicamente de un solo byte)
1046	ASCII/ISO/Windows (bidireccional)	Windows Árabe ST5
1089	ASCII/ISO/Windows (bidireccional)	ISO 8859-6 (Árabe) ST5
1112	EBCDIC de un solo byte	Báltico multilingüe
1122	EBCDIC de un solo byte	Estonio
1123	EBCDIC de un solo byte	Ucrania

1125	ASCII/ISO/Windows	Ucrania
1129	ASCII/ISO/Windows	Vietnamita
1130	EBCDIC de un solo byte	Vietnamita
1131	ASCII/ISO/Windows	Bielorrusia
1132	EBCDIC de un solo byte	Lao
1140	EBCDIC de un solo byte	Estados Unidos y otros (con soporte para Euro)
1141	EBCDIC de un solo byte	Austria, Alemania (con soporte para Euro)
1142	EBCDIC de un solo byte	Dinamarca, Noruega (con soporte para Euro)
1143	EBCDIC de un solo byte	Finlandia, Suecia (con soporte para Euro)
1144	EBCDIC de un solo byte	Italia (con soporte para Euro)
1145	EBCDIC de un solo byte	España, Latinoamérica (con soporte para Euro)
1146	EBCDIC de un solo byte	Reino Unido (con soporte para Euro)
1147	EBCDIC de un solo byte	Francia (con soporte para Euro)
1148	EBCDIC de un solo byte	Latino 1 (MNCS) (soporte para Euro)
1149	EBCDIC de un solo byte	Islandia (con soporte para Euro)
1200	Unicode	Unicode UCS-2 (little-endian)
1250	ASCII/ISO/Windows	Windows Latino 2
1251	ASCII/ISO/Windows	Windows Cirílico
1252	ASCII/ISO/Windows	Windows Latino 1
1253	ASCII/ISO/Windows	Windows Griego
1254	ASCII/ISO/Windows	Windows Turquía
1255	ASCII/ISO/Windows (bidireccional)	Windows Hebreo ST5
1256	ASCII/ISO/Windows (bidireccional)	Windows Árabe ST5
1257	ASCII/ISO/Windows	Windows Báltico
1258	ASCII/ISO/Windows	Windows Vietnam
1364	EBCDIC de byte mixto	Japonés
1388	EBCDIC de byte mixto	Chino simplificado
1399	EBCDIC de byte mixto	Japonés (en V4R5 y superior)
4396	EBCDIC de doble byte	Japonés (subconjunto de 300)
4930	EBCDIC de doble byte	Coreano
4931	EBCDIC de doble byte	Chino tradicional (subconjunto de 835)
4933	EBCDIC de doble byte	Chino simplificado GBK gráfico
4948	ASCII/ISO/Windows	Latino 2 (subconjunto de 852)
4951	ASCII/ISO/Windows	Cirílico (subconjunto de 855)
5026	EBCDIC de byte mixto	Japonés
5035	EBCDIC de byte mixto	Japonés
5123	EBCDIC de un solo byte	Japonés (únicamente de un solo byte, con soporte para Euro)
5351	ASCII/ISO/Windows (bidireccional)	Windows Hebreo (soporte para Euro) ST5
8492	EBCDIC de doble byte	Japonés (subconjunto de 300)
8612	EBCDIC de un solo byte	Árabe EBCDIC ST5
9026	EBCDIC de doble byte	Coreano (subconjunto de 834)
9029	EBCDIC de doble byte	Chino simplificado (subconjunto de 4933)

9066	ASCII/ISO/Windows	Tailandés (SBCS ampliado)
12588	EBCDIC de doble byte	Japonés (subconjunto de 300)
13122	EBCDIC de doble byte	Coreano (subconjunto de 834)
16684	EBCDIC de doble byte	Japonés (disponible en V4R5)
17218	EBCDIC de doble byte	Coreano (subconjunto de 834)
12708	EBCDIC de un solo byte	Árabe EBCDIC ST7
13488	Unicode	Unicode UCS-2 (big-endian)
28709	EBCDIC de un solo byte	Chino tradicional (únicamente de un solo byte)
61952	Unicode	iSeries y AS/400e Unicode (utilizado principalmente en el IFS)
62211	EBCDIC de un solo byte	Hebreo EBCDIC ST5
62224	EBCDIC de un solo byte	Árabe EBCDIC ST6
62235	EBCDIC de un solo byte	Hebreo EBCDIC ST6
62245	EBCDIC de un solo byte	Hebreo EBCDIC ST10

Clase CommandPrompter

La clase CommandPrompter solicita el parámetro en un mandato determinado. CommandPrompter ofrece funciones parecidas a la solicitud de mandatos CL de iSeries (pulsando F4) y las mismas funciones que la solicitud de mandatos de Management Central.

Para utilizar CommandPrompter, el servidor debe ejecutar OS/400 V4R4 o posterior. Para obtener más información, consulte los APAR informativos de iSeries Navigator y vea los arreglos necesarios para el soporte del programa de solicitud de mandatos gráfico.

Para utilizar CommandPrompter también es necesario tener los siguientes archivos jar en la CLASSPATH:

- jt400.jar
- jui400.jar
- util400.jar
- x4j400.jar
- jhall.jar

Todos los archivos jar, excepto jhall.jar, se incluyen en Toolbox para Java. Para obtener más información acerca de los archivos jar de Toolbox para Java, consulte Archivos jar. Para obtener más información sobre cómo bajar jhall.jar, consulte el sitio Web de Sun JavaHelp(TM).

Para construir un objeto CommandPrompter, se le pasan parámetros para el marco padre que inicia el programa de solicitud, el objeto AS400 en el que se solicitará el mandato y la serie del mandato. La serie del mandato puede ser un nombre de mandato, una serie de mandato completa o un nombre de mandato parcial, como por ejemplo crt*.

La visualización de CommandPrompter es un diálogo modal que el usuario debe cerrar antes de volver al marco padre. CommandPrompter maneja los errores encontrados durante la solicitud.

Ejemplo: cómo se utiliza CommandPrompter con las clases CommandCall y AS400Message para solicitar y ejecutar un mandato

RunJavaApplication

Las clases RunJavaApplication y VRunJavaApplication son programas de utilidad que permiten ejecutar programas Java en la máquina virtual Java de iSeries. A diferencia de las clases JavaApplicationCall y VJavaApplicationCall a las que se llama desde el programa Java, las clases RunJavaApplication y VRunJavaApplication son programas completos.

La clase RunJavaApplication es un programa de utilidad de línea de mandatos. Permite establecer el entorno (por ejemplo, la variable CLASSPATH y las propiedades) del programa Java. Primero se especifica el nombre del programa Java y sus parámetros y después se inicia el programa. Una vez iniciado, se puede enviar una entrada al programa Java, el cual la recibe por medio de la entrada estándar. El programa Java escribe la salida en la salida estándar y en la salida de error estándar.

El programa de utilidad VRunJavaApplication tiene las mismas posibilidades. La diferencia es que VJavaApplicationCall utiliza una interfaz gráfica de usuario, mientras que JavaApplicationCall es una interfaz de línea de mandatos.

JPing

La clase JPing es un programa de utilidad de la línea de mandatos que permite consultar los servidores para ver qué servicios están en ejecución y qué puertos están en servicio. Para consultar los servidores desde dentro de una aplicación Java, utilice la clase AS400JPing.

En el javadoc de JPing encontrará más información acerca de cómo se utiliza JPing desde dentro de una aplicación Java.

Para efectuar una llamada a JPing desde la línea de mandatos, utilice la sintaxis que se indica a continuación:

```
java utilities.JPing Sistema [opciones]
```

donde:

- Sistema = el servidor iSeries que desea consultar
- [opciones] = una o varias de las opciones disponibles

Opciones

Puede utilizar una o varias de las opciones siguientes. En el caso de las opciones que tienen abreviaturas, la abreviatura se indica entre paréntesis.

-help (-h o **-?)**

Visualiza el texto de ayuda.

-service *Servicio_OS/400* **(-s** *Servicio_OS/400***)**

Especifica un servicio específico para realizar un ping. La acción por omisión consiste en realizar un ping de todos los servicios. Puede utilizar esta opción para especificar uno de los servicios siguientes: as-file, as-netprt, as-rmtcmd, as-dtaq, as-database, as-ddm, as-central y as-signon.

-ssl

Especifica si se realizará un ping de los puertos ssl o no. La acción por omisión consiste en no realizar un ping de los puertos ssl.

-timeout (-t)

Especifica el tiempo de espera en milisegundos. El valor por omisión es 20000 (o 20 segundos).

Ejemplo: cómo se utiliza JPing desde la línea de mandatos

Por ejemplo, utilice el mandato siguiente para realizar un ping del servicio as-dtaq incluyendo los puertos ssl y con un tiempo de espera de 5 segundos:

```
java utilities.JPing myServer -s as-dtaq -ssl -t 5000
```

Clases de vaccess

IBM Toolbox para Java proporciona un conjunto de clases de GUI (interfaz gráfica de usuario) en el paquete vaccess. Estas clases utilizan las clases de acceso para recuperar los datos y presentárselos al usuario.

Los programas Java que utilizan las clases de vaccess de IBM Toolbox para Java necesitan Swing 1.1. Para obtener Swing 1.1, ejecute Java 2 o baje Swing 1.1 de Sun Microsystems, Inc.. En los releases pasados, IBM Toolbox para Java requería Swing 1.0.3; V4R5 es el primer release en el que está soportado Swing 1.1. Para pasar a Swing 1.1, se hicieron algunos cambios de programación; por lo tanto, usted también puede que tenga que hacer algunos cambios de programación. Consulte la página de Sun Microsystems, Inc. JFC para obtener más información acerca de Swing.

Si desea más información acerca de las relaciones entre las clases de GUI de IBM Toolbox para Java, las clases de acceso y Java Swing, consulte el diagrama de clases de vaccess.

Utilice las clases de secciones de panel AS400 para visualizar los datos de iSeries.

Hay interfaces API disponibles para acceder a los siguientes recursos de iSeries y a sus correspondientes herramientas:

- Llamada a mandato
- Colas de datos
- Eventos de error*
- Sistema de archivos integrado
- JavaApplicationCall
- JDBC
- Trabajos*
- Mensajes*
- Permiso
- Imprimir* incluido el visor de archivos en spool
- ProgramCall y ProgramParameter
- Acceso a nivel de registro
- Listas de recursos
- Estado del sistema
- Valores del sistema
- Usuarios y grupos

Nota: las secciones de panel AS400 se utilizan junto con otras clases de vaccess (vea los elementos marcados con un asterisco, más arriba) para presentar los recursos de iSeries y permitir su manipulación.

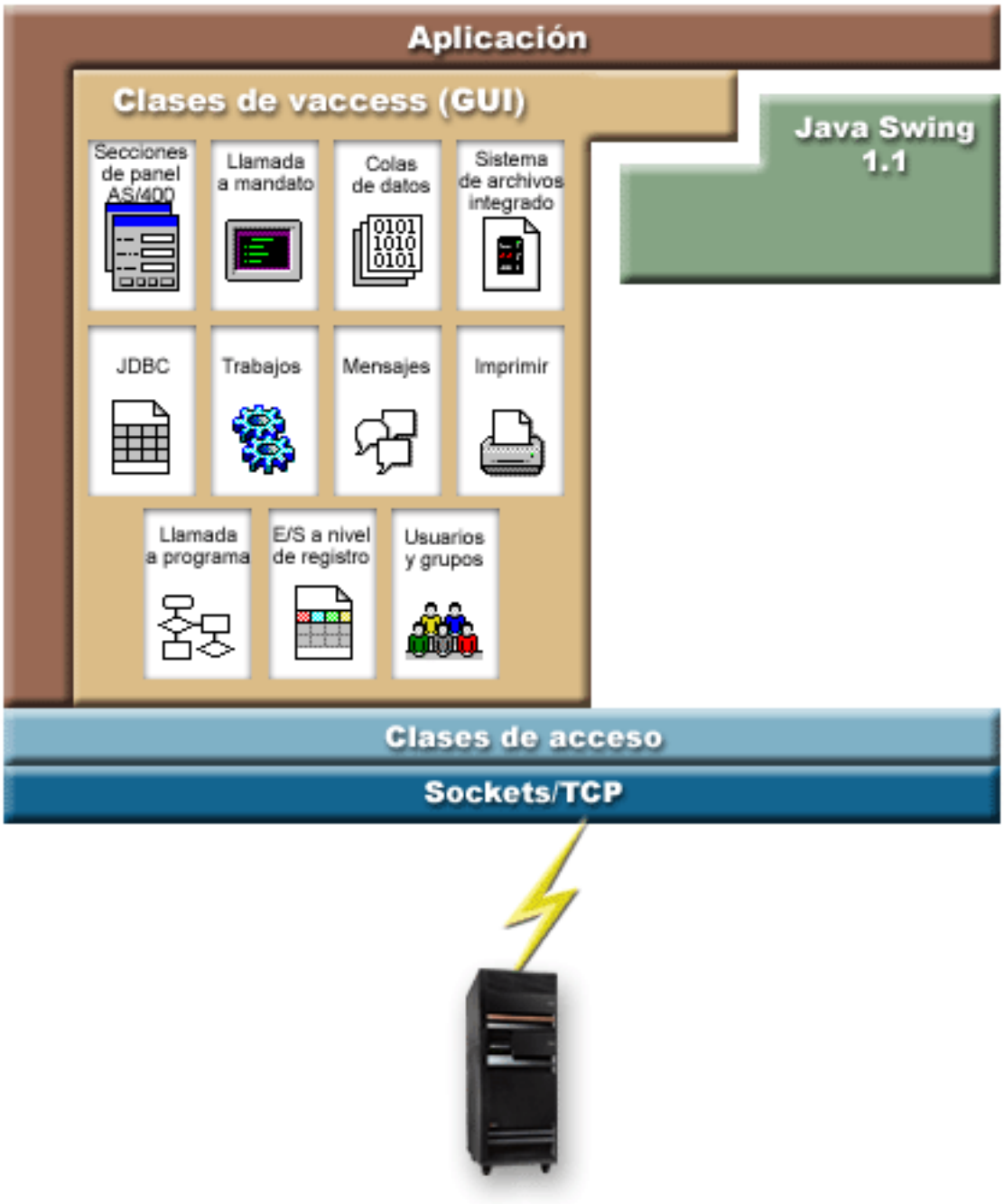
Al programar con los componentes de la interfaz gráfica de usuario de IBM Toolbox para Java, utilice las clases de eventos de error para informar al usuario de los eventos de error y permitir su manejo.

En Clases de acceso encontrará más información sobre cómo se accede a los datos de iSeries.

Clases de vaccess

IBM Toolbox para Java proporciona clases de interfaz gráfica de usuario (GUI) en el paquete vaccess que permiten recuperar, visualizar y, en algunos casos, manipular los datos del servidor. Estas clases utilizan la infraestructura Java Swing 1.1. La figura 1 muestra la relación que hay entre estas clases:

Figura 1: clases de vaccess



AS400Panes

Las secciones de panel AS/400 (AS400Pane) son componentes del paquete vaccess que presentan uno o varios recursos de servidor de una GUI y permiten su manipulación. El comportamiento de cada recurso de servidor varía en función del tipo de recurso.

Todas las secciones amplían la clase Component Java. Como consecuencia, pueden añadirse a cualquier marco, ventana o contenedor AWT.

Los objetos AS400Pane disponibles son los siguientes:

- AS400DetailsPane: presenta una lista de recursos de servidor en una tabla en la que cada fila visualiza diversos detalles acerca de un recurso individual. La tabla permite realizar la selección de uno o varios recursos.
- AS400ExplorerPane: combina una sección AS400TreePane y una sección AS400DetailsPane para que el recurso seleccionado en el árbol se muestre en la sección de detalles.
- AS400JDBCDataSourcePane: presenta los valores de propiedad de un objeto AS400JDBCDataSource.
- AS400ListPane: presenta una lista de recursos de servidor y permite realizar la selección de uno o varios recursos.
- AS400TreePane: presenta una jerarquía en forma de árbol de los recursos de servidor y permite realizar la selección de uno o varios recursos.

Recursos de servidor

Los recursos de servidor se representan en la interfaz gráfica de usuario mediante un icono y un texto. Los recursos de servidor están definidos con relaciones jerárquicas, por las que un recurso puede tener un padre y cero o más hijos. Se trata de relaciones predefinidas que permiten especificar qué recursos se visualizan en un objeto AS400Pane. Por ejemplo, VJobList es el padre de cero o más objetos VJob y esta relación jerárquica se representa gráficamente en un objeto AS400Pane.

IBM Toolbox para Java proporciona acceso a los siguientes recursos de servidor:

- VIFSDirectory representa un directorio en el sistema de archivos integrado.
- VJob y VJobList representan, respectivamente, un trabajo o una lista de trabajos.
- VMessageList y VMessageQueue representan, respectivamente, una lista de mensajes devueltos desde una llamada a mandato (CommandCall) o a programa (ProgramCall) o una cola de mensajes.
- VPrinter, VPrinters y VPrinterOutput representan, respectivamente, una impresora, una lista de impresoras o una lista de archivos en spool.
- VUserList representa una lista de usuarios.

Todos los recursos son implementaciones de la interfaz VNode.

Establecimiento de la raíz

Para especificar qué recursos de servidor se presentan en un objeto AS400Pane, establezca la raíz mediante el constructor o el método setRoot(). La raíz define el objeto de nivel superior y su uso varía en función de la sección:

- AS400ListPane: presenta todos los hijos de la raíz en su lista.
- AS400DetailsPane: presenta todos los hijos de la raíz en su tabla.
- AS400TreePane: utiliza la raíz como raíz de su árbol.
- AS400ExplorerPane: utiliza la raíz como raíz de su árbol.

Es posible hacer cualquier combinación de secciones y raíces.

En el ejemplo siguiente se crea una sección AS400DetailsPane para presentar la lista de usuarios definidos en el sistema:

```
        // Cree el recurso de servidor que
        // representa una lista de usuarios.
        // Suponga que "system" es un objeto
        // AS400 creado e inicializado
        // en otra parte.
    VUserList userList = new VUserList (system);

        // Cree el objeto AS400DetailsPane
        // y establezca que su raíz sea la
        // lista de usuarios.
    AS400DetailsPane detailsPane = new AS400DetailsPane ();
    detailsPane.setRoot (userList);
```

```
// Añada la sección de detalles a un marco.
// Suponga que "frame" es un objeto JFrame
// creado en otra parte.
 frame.getContentPane ().add (detailsPane);
```

Carga del contenido

Los objetos AS400Pane y los objetos de recurso de servidor, cuando se crean, se inicializan en un estado por omisión. La información relevante que constituye el contenido de la sección no se carga en el momento de la creación de dicha sección.

Para cargar el contenido, la aplicación debe llamar explícitamente al método load(). En la mayoría de los casos, ello hace que se inicie la comunicación con el servidor para recopilar la información relevante. Debido a que, a veces, la tarea de recopilar esta información quizá tarde bastante en llevarse a cabo, la aplicación puede controlar exactamente cuándo se produce esta carga. Por ejemplo, es posible:

- Cargar el contenido antes de añadir la sección a un marco. El marco no aparece hasta que se haya cargado toda la información.
- Cargar el contenido después de añadir la sección a un marco y de visualizar el marco. El marco aparece, pero no contiene mucha información. Se muestra un "cursor de espera" y la información se va rellenando a medida que se carga.

El siguiente ejemplo explica cómo se carga el contenido de una sección de detalles antes de añadirla a un marco:

```
      // Cargue el contenido de la sección de
      // detalles. Supongamos que el objeto detailsPane
      // se ha creado e inicializado
      // en otra parte.
         detailsPane.load ();

// Añada la sección de detalles a un marco.
// Suponga que "frame" es un objeto JFrame
// creado en otra parte.
 frame.getContentPane ().add (detailsPane);
```

Secciones de acciones y propiedades

En tiempo de ejecución, el usuario puede seleccionar un menú emergente en cualquier recurso de servidor. El menú emergente presenta una lista de acciones relevantes que están disponibles para el recurso. Cuando el usuario selecciona una acción en el menú emergente, ésta se realiza. Para cada recurso hay definidas distintas acciones.

En algunos casos, el menú emergente también presenta un elemento que permite al usuario ver una sección de propiedades. Esta sección muestra diversos detalles acerca del recurso y permite al usuario realizar cambios en dichos detalles.

La aplicación puede controlar si las secciones de acciones y propiedades están disponibles utilizando el método setAllowActions() en la sección.

Modelos

Los objetos AS400Pane se implementan mediante el paradigma controlador de modelos-vistas, en el que los datos y la interfaz de usuario están separados en distintas clases. Los objetos AS400Pane integran los modelos de IBM Toolbox para Java con los componentes de la interfaz gráfica de usuario Java. Los modelos gestionan los recursos de servidor y los componentes de vaccess los visualizan gráficamente y manejan la interacción del usuario.

Los objetos AS400Pane proporcionan suficientes funciones para la mayoría de las necesidades. No obstante, una aplicación, si necesita un mayor control del componente JFC, puede acceder directamente a un modelo de servidor y proporcionar una integración personalizada con un componente de vaccess distinto.

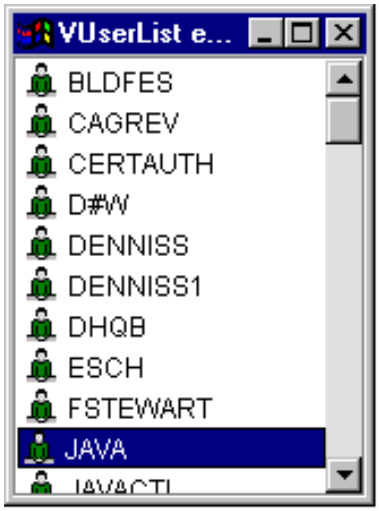
Los modelos disponibles son los siguientes:

- AS400ListModel, que implementa la interfaz ListModel JFC como una lista de recursos de servidor. Puede utilizarse con un objeto JList JFC.
- AS400DetailsModel, que implementa la interfaz TableModel JFC como una tabla de recursos de servidor en la que cada fila contiene diversos detalles acerca de un único recurso. Puede utilizarse con un objeto JTable de JFC.
- AS400TreeModel, que implementa la interfaz TreeModel JFC como un árbol jerárquico de recursos de servidor. Puede utilizarse con un objeto JTree JFC.

Ejemplos

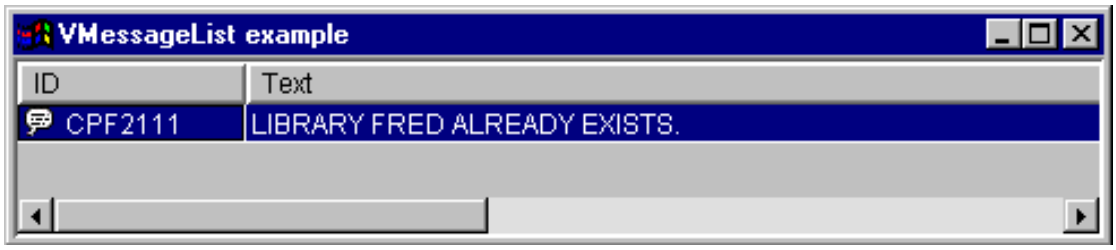
- Presentar una lista de usuarios existentes en el sistema utilizando una sección AS400ListPane con un objeto VUserList. La figura 1 muestra el producto acabado:

  Figura 1: cómo se utiliza AS400ListPane con un objeto VUserList



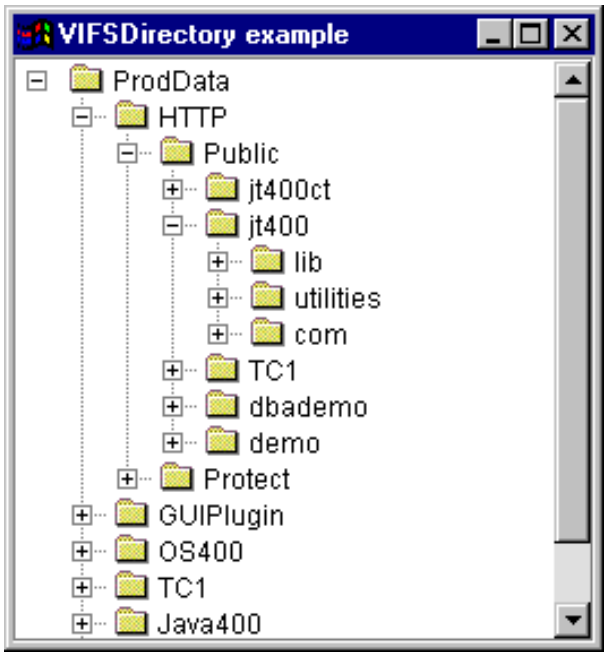
- Presentar la lista de mensajes generados por una llamada a mandato utilizando una sección AS400DetailsPane con un objeto VMessageList.La figura 2 muestra el producto acabado:

  Figura 2: cómo se utiliza AS400DetailsPane con un objeto VMessageList



- Presentar una jerarquía de directorios del sistema de archivos integrado utilizando una sección AS400TreePane con un objeto VIFSDirectory.La figura 3 muestra el producto acabado:

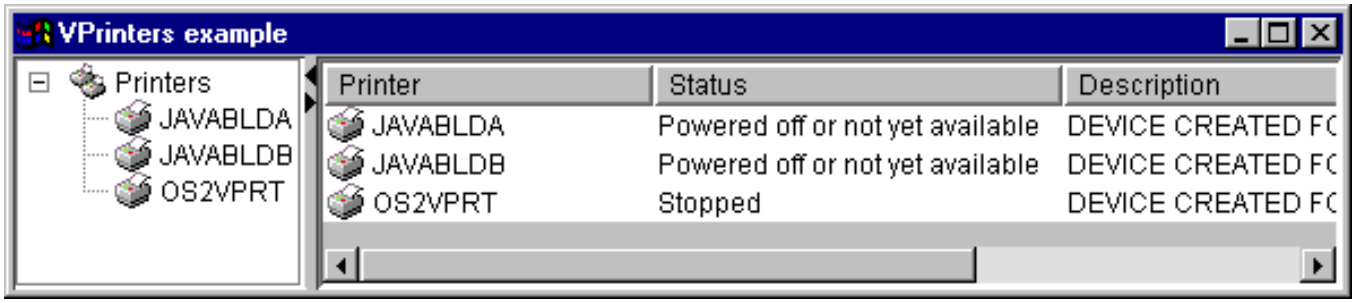
  Figura 3: cómo se utiliza AS400TreePane con un objeto VIFSDirectory



- Presentar recursos de impresión utilizando una sección AS400ExplorerPane con un objeto VPrinters.La figura 4 muestra el producto

acabado:

Figura 4: cómo se utiliza AS400ExplorerPane con un objeto VPrinters



Llamada a mandato

Los componentes de vaccess (GUI) de llamada a mandato permiten a un programa Java presentar un botón o un elemento de menú que llama a un mandato no interactivo del servidor.

Un objeto CommandCallButton representa un botón que, cuando se pulsa, llama a un mandato del servidor. La clase CommandCallButton amplía la clase JButton JFC (clases Java fundamentales) para que todos los botones tengan un aspecto y un comportamiento coherentes.

De forma parecida, un objeto CommandCallMenuItem representa un elemento de menú que, cuando se selecciona, llama a un mandato del servidor. La clase CommandCallMenuItem amplía la clase JMenuItem JFC para que todos los elementos de menú tengan también un aspecto y un comportamiento coherentes.

Para utilizar un componente de la interfaz gráfica de usuario de la llamada a mandato, establezca las propiedades system y command. Estas propiedades se pueden establecer mediante un constructor o con los métodos setSystem() y setCommand().

El siguiente ejemplo crea un botón CommandCallButton. El botón, cuando se pulsa en tiempo de ejecución, crea una biblioteca llamada "FRED":

```
        // Cree el objeto CommandCallButton.
        // Supongamos que "system" es
        // un objeto AS400 creado e
        // inicializado en otra parte. El texto
        // del botón dice "Pulse aquí", y no hay
        // ningún icono.
  CommandCallButton button = new CommandCallButton ("Pulse aquí", null,
system);

        // Establezca el mandato que el botón va a ejecutar.
  button.setCommand ("CRTLIB FRED");

        // Añada el botón a un marco. Supongamos
        // que "frame" es un objeto JFrame
        // creado en otra parte.
    frame.getContentPane ().add (button);
```

Un mandato del servidor, cuando se ejecuta, puede devolver cero o más mensajes del servidor. Para detectar cuándo se ejecuta el mandato del servidor, añada un escuchador ActionCompletedListener al botón o al elemento de menú utilizando el método addActionCompletedListener(). El mandato, cuando se ejecuta, activa un evento ActionCompletedEvent para todos estos escuchadores. Un escuchador puede utilizar el método getMessageList() para recuperar los mensajes del servidor que el mandato haya generado.

Este ejemplo añade un escuchador ActionCompletedListener que procesa todos los mensajes del servidor generados por el mandato:

```
        // Añada un ActionCompletedListener que
        // se implementa utilizando una clase
        // interna anónima. Es un modo práctico
        // de especificar escuchadores de
        // eventos simples.
    button.addActionCompletedListener (new ActionCompletedListener ()
    {
         public void actionCompleted (ActionCompletedEvent event)
         {
        // Convierta el origen del evento en un
        // CommandCallButton.
```

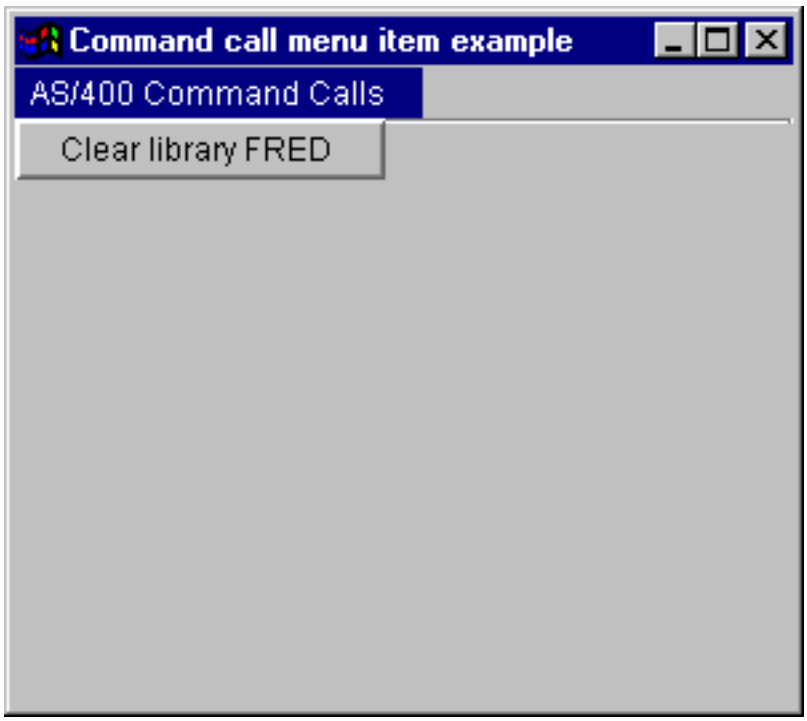
```
                CommandCallButton sourceButton = (CommandCallButton)
event.getSource ();

          // Obtenga la lista de mensajes del servidor
          // que el mandato ha generado.
                AS400Message[] messageList = sourceButton.getMessageList ();

          // ...Procese la lista de mensajes.
           }
     });
```

Ejemplos

Este ejemplo muestra cómo se utiliza un elemento de menú de llamada a mandato (CommandCallMenuItem) en una aplicación.

La figura 1 muestra el componente de la interfaz gráfica de usuario CommandCall:

Figura 1: componente GUI CommandCall



Colas de datos

Los componentes gráficos de cola de datos permiten a un programa Java utilizar cualquier componente gráfico de texto JFC (clases Java fundamentales) para leer o escribir en una cola de datos del servidor.

Las clases DataQueueDocument y KeyedDataQueueDocument son implementaciones de la interfaz Document JFC. Estas clases se pueden utilizar directamente con cualquier componente gráfico de texto JFC. En JFC hay disponibles varios componentes de texto, como por ejemplo, campos de una sola línea (JTextField) y áreas de texto de varias líneas (JTextArea).

Los documentos de cola de datos asocian el contenido de un componente de texto a una cola de datos del servidor. (Un componente de texto es un componente gráfico que permite visualizar texto que el usuario puede editar opcionalmente.) El programa Java puede realizar operaciones de leer y escribir entre el componente de texto y la cola de datos, en cualquier momento. Utilice el documento DataQueueDocument para las colas de datos **secuenciales**, y el documento KeyedDataQueueDocument para las colas de datos **por clave**.

Para utilizar un documento DataQueueDocument, establezca las propiedades system y path. Estas propiedades se pueden establecer mediante un constructor o con los métodos setSystem() y setPath(). El objeto DataQueueDocument se "conecta" entonces al componente de texto, utilizando habitualmente el constructor o el método setDocument() del componente de texto. Los documentos KeyedDataQueueDocument funcionan de idéntico modo.

El siguiente ejemplo crea un DataQueueDocument cuyo contenido se asocia a una cola de datos:

```
         // Cree el objeto DataQueueDocument.
         // Suponga que "system" es
         // un objeto AS400 creado e
         // inicializado en otra parte.
     DataQueueDocument dqDocument = new DataQueueDocument (system,
"/QSYS.LIB/MYLIB.LIB/MYQUEUE.DTAQ");

         // Cree un área de texto para presentar
         // el documento.
     JTextArea textArea = new JTextArea (dqDocument);

         // Añada el área de texto a un marco.
         // Suponga que "frame" es un objeto JFrame
    // creado en otra parte.
     frame.getContentPane ().add (textArea);
```

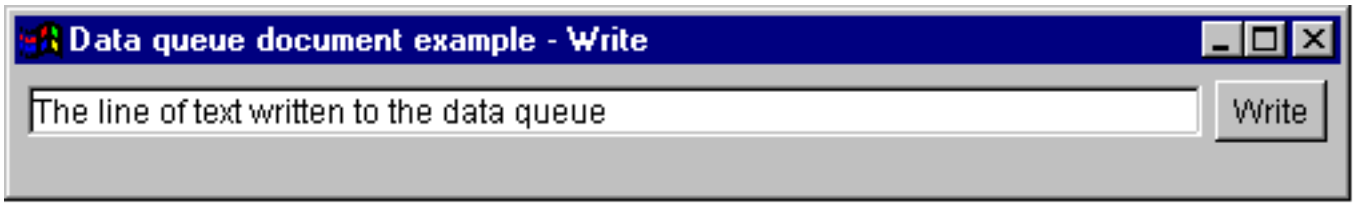
Inicialmente, el contenido del componente de texto está vacío. Utilice el método read() o peek() para rellenar el contenido con la próxima entrada de la cola. Utilice write() para escribir el contenido del componente de texto en la cola de datos. Tenga en cuenta que estos documentos sólo funcionan con entradas de cola de datos de tipo serie.

Ejemplos

Ejemplo de cómo se utiliza un documento de cola de datos (DataQueueDocument) en una aplicación.

La figura 1 muestra el componente de la interfaz gráfica de usuario DataQueueDocument que se está utilizando en un JTextField. Se ha añadido un botón para proporcionar una interfaz GUI que permita al usuario escribir el contenido del campo de texto en la cola de datos.

Figura 1: componente GUI DataQueueDocument


Data queue document example - Write
The line of text written to the data queue
Write

Eventos de error

En la mayoría de los casos, los componentes de la interfaz gráfica de usuario (GUI) de IBM Toolbox para Java activan eventos de error, en vez de lanzar excepciones.

Un evento de error es un objeto que envuelve una excepción lanzada por un componente interno.

Puede proporcionar un escuchador de errores que maneje todos los eventos de error activados por un determinado componente de la interfaz gráfica de usuario. Siempre que se lanza una excepción, se llama al escuchador y éste puede proporcionar la debida notificación del error. Por omisión, no se lleva a cabo ninguna acción cuando se activan eventos de error.

IBM Toolbox para Java proporciona un componente de interfaz gráfica de usuario denominado ErrorDialogAdapter, que automáticamente muestra un diálogo al usuario cada vez que se activa un evento de error.

Ejemplos

Los ejemplos que hay a continuación muestran cómo se manejan los errores y se define un escuchador de errores simple.

Ejemplo: manejar los eventos de error visualizando un diálogo

El ejemplo que sigue muestra cómo se pueden manejar los eventos de error visualizando un diálogo:

```
        // ...Ya se ha terminado todo el trabajo
        // de diseño de un componente de la
        // interfaz gráfica de usuario. Ahora, añada
        // al componente un ErrorDialogAdapter que
        // haga de escuchador. Éste informará de
        // todos los eventos de error activados por
        // ese componente, mediante la visualización
        // de un diálogo.
    ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (parentFrame);
    component.addErrorListener (errorHandler);
```

Puede escribir un escuchador de errores personalizado que maneje los errores de otro modo. Para ello, utilice la interfaz ErrorListener.

Ejemplo: definir un escuchador de errores

El ejemplo siguiente muestra cómo se define un escuchador de errores simple, que únicamente imprime los errores en System.out:

```
    class MyErrorHandler
    implements ErrorListener
    {
        // Se invoca este método siempre
        // que se activa un evento de error.
            public void errorOccurred(ErrorEvent event)
            {
                 Exception e = event.getException ();
          System.out.println ("Error: " + e.getMessage ());
            }
    }
```

Ejemplo: manejar los eventos de error utilizando un escuchador de errores

El ejemplo siguiente muestra cómo se manejan los eventos de error para un componente de la interfaz gráfica de usuario que utilice este manejador personalizado:

```
MyErrorHandler errorHandler = new MyErrorHandler ();
component.addErrorListener (errorHandler);
```

Sistema de archivos integrado

Los componentes de la interfaz gráfica de usuario del sistema de archivos integrado permiten a un programa Java presentar en una GUI los directorios y los archivos del sistema de archivos integrado del servidor.

Los componentes disponibles son los siguientes:

- IFSFileDialog: presenta un diálogo que permite al usuario elegir un directorio y seleccionar un archivo, navegando por la jerarquía de directorios.
- VIFSDirectory: es un recurso que representa un directorio del sistema de archivos integrado para utilizarse en los objetos AS400Pane.
- IFSTextFileDocument: representa un archivo de texto que se utiliza en cualquier componente gráfico de texto JFC (clases Java fundamentales).

Para utilizar los componentes de la interfaz gráfica de usuario del sistema de archivos integrado, establezca la propiedad system y la propiedad path o directory. Estas propiedades se pueden establecer mediante un constructor o con los métodos setDirectory() (para IFSFileDialog) o setSystem() y setPath() (para VIFSDirectory y IFSTextFileDocument).

La vía de acceso debe establecerse en un valor que no sea "/QSYS.LIB", porque este directorio suele ser de gran tamaño y la tarea de bajar su contenido puede ser de larga duración.

Diálogos de archivo

La clase IFSFileDialog es un diálogo que permite al usuario recorrer los directorios del sistema de archivos integrado del servidor y seleccionar un archivo. El llamador puede establecer el texto de los botones del diálogo. Además, el llamador puede utilizar objetos FileFilter, que permiten al usuario limitar las opciones a ciertos archivos.

Si el usuario selecciona un archivo en el diálogo, utilice el método getFileName() para obtener el nombre del archivo seleccionado. Utilice el método getAbsolutePath() para obtener el nombre completo de la vía de acceso del archivo seleccionado.

En el siguiente ejemplo se define un diálogo de archivo del sistema de archivos integrado con dos filtros de archivo:

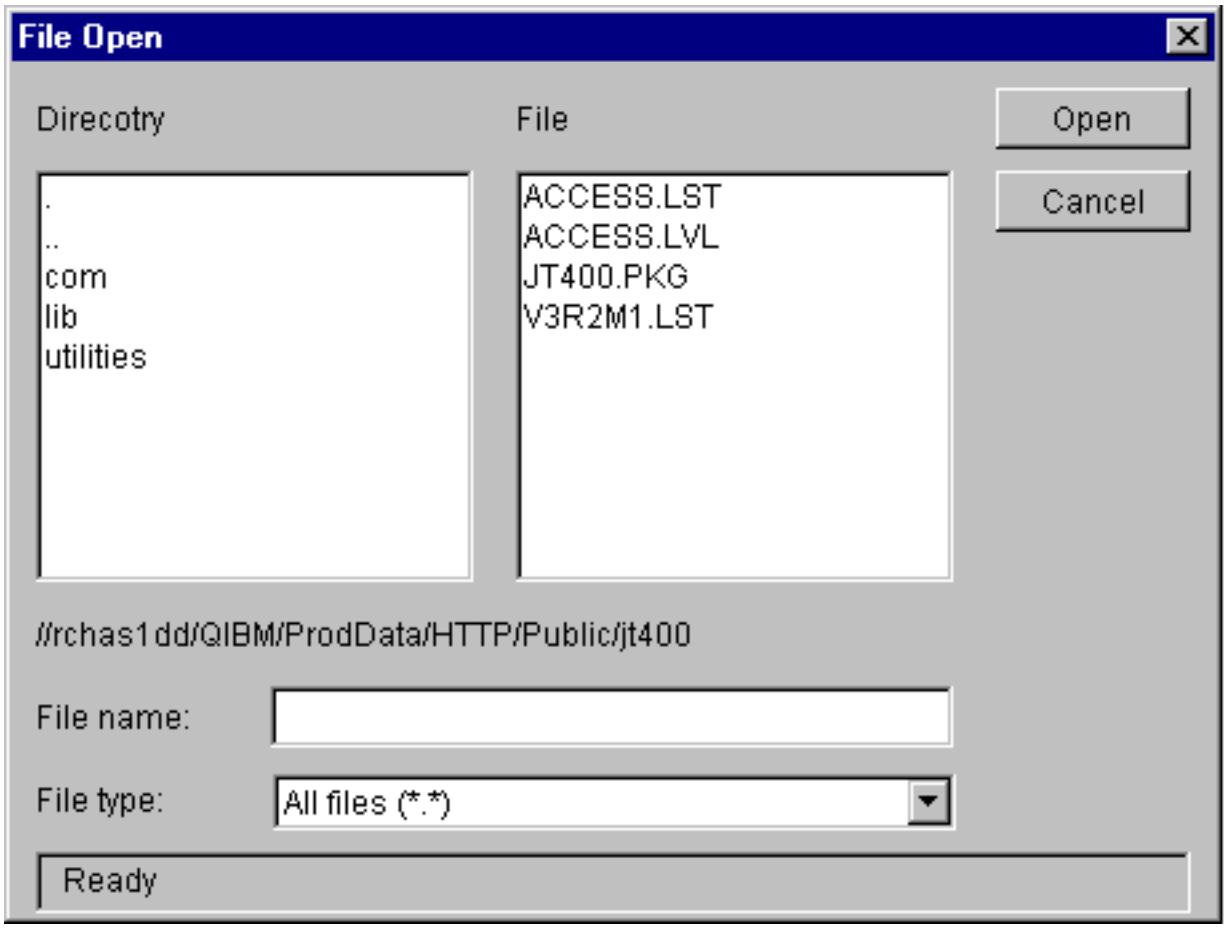
```
         // Cree un objeto IFSFileDialog
         // estableciendo el texto de la barra de título.
         // Suponga que "system" es un objeto AS400
         // y "frame" es un objeto JFrame
         // creados e inicializados en otra parte.
     IFSFileDialog dialog = new IFSFileDialog (frame, "Seleccionar un
archivo", system);

         // Establezca una lista de filtros para el diálogo.
         // Se utilizará el primer filtro cuando
         // el diálogo se visualice por primera vez.
     FileFilter[] filterList = {new FileFilter ("Todos los archivos (*.*)",
"*.*"),
                                new FileFilter ("Archivos HTML (*.HTML",
"*.HTM")};
         // Luego, establezca los filtros en el diálogo.
     dialog.setFileFilter (filterList, 0);

         // Establezca el texto en los botones.
     dialog.setOkButtonText ("Abrir");
     dialog.setCancelButtonText ("Cancelar");

         // Muestre el diálogo. Si el usuario
         // seleccionó un archivo pulsando el botón
         // "Abrir", imprima la
         // vía de acceso del archivo seleccionado.
     if (dialog.showDialog () == IFSFileDialog.OK)
          System.out.println (dialog.getAbsolutePath ());
```

Ejemplo

Presentar un IFSFileDialog e imprimir la selección, si es que hay alguna.

La figura 1 muestra el componente de la interfaz gráfica de usuario IFSFileDialog:

Figura 1: componente GUI IFSFileDialog


File Open
Direcotry
File
Open
Cancel
.
..
com
lib
utilities
ACCESS.LST
ACCESS.LVL
JT400.PKG
V3R2M1.LST
//rchas1dd/QIBM/ProdData/HTTP/Public/jt400
File name:
File type:
All files (*.*)
Ready

Directorios en objetos AS400Pane

Las secciones de panel AS/400 (AS400Pane) son componentes de la interfaz gráfica de usuario que presentan uno o varios recursos del servidor y permiten su manipulación. Un objeto VIFSDirectory es un recurso que representa un directorio del sistema de archivos integrado para utilizarse en los objetos AS400Pane. Los objetos AS400Pane y VIFSDirectory se pueden utilizar conjuntamente para presentar muchas vistas del sistema de archivos integrado, y para permitir al usuario navegar, manipular y seleccionar directorios y archivos.

Para utilizar un directorio VIFSDirectory, establezca las propiedades system y path. Estas propiedades se establecen mediante un constructor o con los métodos setSystem() y setPath(). A continuación puede conectar el objeto VIFSDirectory al objeto AS400Pane para que haga de raíz, utilizando el constructor o el método setRoot() del objeto AS400Pane.

VIFSDirectory dispone de otras propiedades que son de utilidad para definir el conjunto de directorios y archivos presentados en los objetos AS400Pane. Utilice setInclude() para especificar si han de aparecer los directorios, los archivos, o las dos cosas. Utilice setPattern() para establecer un filtro en los elementos que se muestran, especificando un patrón con el que deba coincidir el nombre del archivo. En los patrones se pueden emplear caracteres comodín, tales como "*" y "?". De modo semejante, utilice setFilter() para establecer un filtro con un objeto IFSFileFilter.

Los objetos AS400Pane y VIFSDirectory, cuando se crean, se inicializan en un estado por omisión. Los subdirectorios y los archivos que constituyen el contenido del directorio raíz no se han cargado. Para cargar el contenido, el llamador debe llamar explícitamente al método load() en cualquiera de los dos objetos para iniciar la comunicación con el servidor a fin de recopilar el contenido del directorio.

En tiempo de ejecución, un usuario puede llevar a cabo acciones en cualquier directorio o archivo pulsando sobre él con el botón derecho del ratón para visualizar el menú de contexto. El menú de contexto de los directorios puede contener los elementos siguientes:

- **Crear archivo** - Crea un archivo en el directorio. Esta acción dará al archivo un nombre por omisión.
- **Crear directorio** - Crea un subdirectorio con un nombre por omisión.
- **Redenominar** - Redenomina un directorio.
- **Suprimir** - Suprime un directorio.
- **Propiedades** - Visualiza propiedades tales como la ubicación, el número de archivos y subdirectorios y la fecha de modificación.

El menú de contexto de los archivos puede contener los elementos siguientes:

- **Editar** - Edita un archivo de texto en otra ventana.
- **Ver** - Visualiza un archivo de texto en otra ventana.
- **Redenominar** - Redenomina un archivo.
- **Suprimir** - Suprime un archivo.
- **Propiedades** - Visualiza propiedades tales como la ubicación, el tamaño, la fecha de modificación y los atributos.

Los usuarios solo pueden leer o escribir en los directorios y archivos sobre los que poseen autorización. Además, el llamador puede impedir que el usuario lleve a cabo acciones utilizando para ello el método setAllowActions() en la sección.

En el ejemplo siguiente se crea un directorio VIFSDirectory y se presenta en una sección AS400ExplorerPane:

```
         // Cree el objeto VIFSDirectory.
         // Suponga que "system" es un objeto
         // AS400 creado e inicializado
         // en otra parte.
     VIFSDirectory root = new VIFSDirectory (system,
"/DirectoryA/DirectoryB");
```

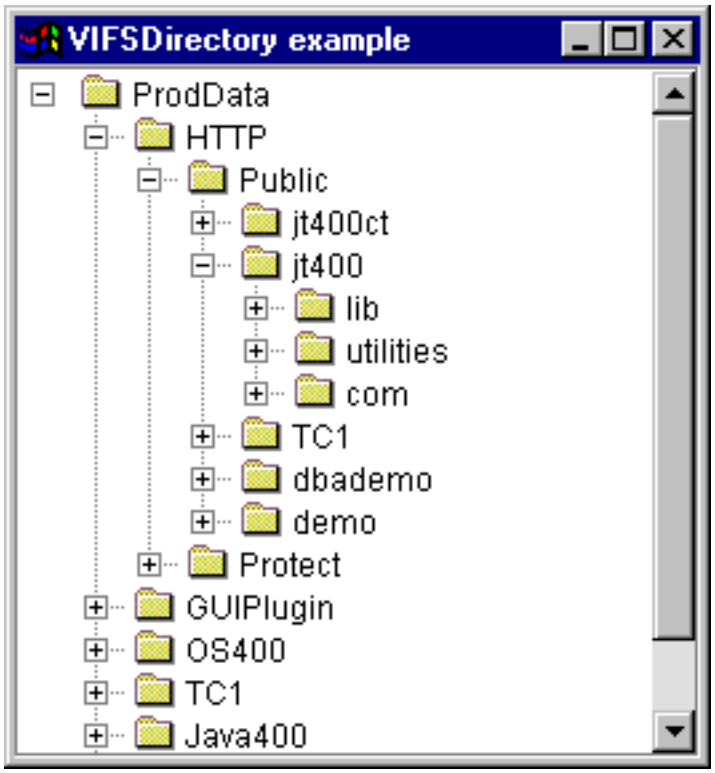
```
    // Cree y cargue un objeto AS400ExplorerPane.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
               explorerPane.load ();

    // Añada la sección de panel explorador a un marco.
    // Suponga que "frame" es un objeto JFrame
    // creado en otra parte.
frame.getContentPane ().add (explorerPane);
```

Ejemplo

Presentar una jerarquía de directorios del sistema de archivos integrado utilizando una sección AS400TreePane con un objeto VIFSDirectory.

La figura 1 muestra el componente de la interfaz gráfica de usuario VIFSDirectory:

Figura 1: componente GUI VIFSDirectory



IFSTextFileDocument

Los documentos de archivo de texto permiten a un programa Java utilizar cualquier componente gráfico de texto JFC (clases Java fundamentales) para editar o ver archivos de texto del sistema de archivos integrado de un servidor. (Un componente de texto es un componente gráfico que permite visualizar texto que el usuario puede editar opcionalmente.)

La clase IFSTextFileDocument es una implementación de la interfaz Document JFC. Esta clase se puede utilizar directamente con cualquier componente gráfico de texto JFC. En JFC hay disponibles varios componentes de texto, como por ejemplo, campos de una sola línea (JTextField) y áreas de texto de varias líneas (JTextArea).

Los documentos de archivo de texto asocian el contenido de un componente de texto a un archivo de texto. El programa Java puede realizar operaciones de cargar y guardar entre el componente de texto y el archivo de texto en cualquier momento.

Para utilizar un documento IFSTextFileDocument, establezca las propiedades system y path. Estas propiedades se pueden establecer mediante un constructor o con los métodos setSystem() y setPath(). El objeto IFSTextFileDocument se "conecta" entonces al componente de texto, utilizando habitualmente el constructor o el método setDocument() del componente de texto.

Inicialmente, el contenido del componente de texto está vacío. Utilice load() para cargar el contenido del archivo de texto. Utilice save() para guardar el contenido del componente de texto en el archivo de texto.

En el ejemplo siguiente, se crea y se carga un documento IFSTextFileDocument:

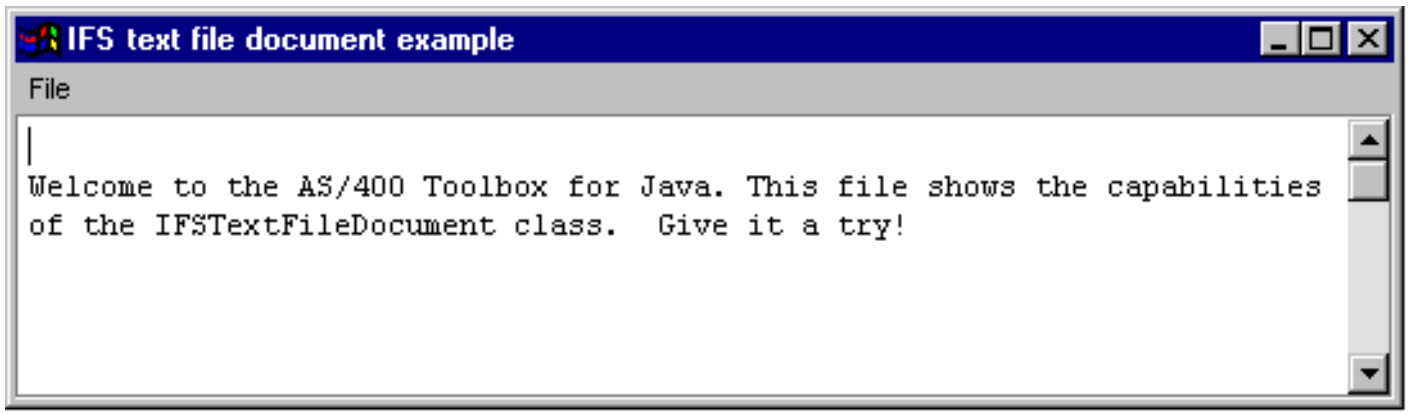
```
        // Cree y cargue un objeto
        // IFSTextFileDocument. Supongamos
        // que "system" es un objeto AS400
        // creado e inicializado en otra parte.
     IFSTextFileDocument ifsDocument = new IFSTextFileDocument (system,
"/DirectoryA/MyFile.txt");
     ifsDocument.load ();

        // Cree un área de texto para presentar
        // el documento.
     JTextArea textArea = new JTextArea (ifsDocument);

        // Añada el área de texto a un marco.
        // Suponga que "frame" es un objeto JFrame
        // creado en otra parte.
     frame.getContentPane ().add (textArea);
```

Ejemplo

Presentar un IFSTextFileDocument en una sección JTextPane.

La figura 1 muestra el componente de la interfaz gráfica de usuario IFSTextFileDocument:

Figura 1: ejemplo de documento de archivo de texto de IFS


IFS text file document example
File
Welcome to the AS/400 Toolbox for Java. This file shows the capabilities
of the IFSTextFileDocument class. Give it a try!

Clase VJavaApplicationCall

La clase VJavaApplicationCall le permite ejecutar en el servidor una aplicación Java desde un cliente utilizando una interfaz gráfica de usuario (GUI).

La GUI es un panel que consta de dos secciones. La sección superior es una ventana que visualiza la salida que el programa Java escribe en la salida estándar y en la salida de errores estándar. La sección inferior es un campo de entrada en el que el usuario escribe el entorno Java, el programa Java que se ha de ejecutar con los parámetros, y la entrada que el programa Java recibe por medio de la entrada estándar. En Opciones de mandato Java encontrará más información.

Por ejemplo, este código crearía la siguiente GUI para el programa Java.

VJavaApplicationCall es una clase a la que se llama desde el programa Java. Sin embargo, IBM Toolbox para Java también proporciona un programa de utilidad que es una aplicación Java completa que se puede utilizar para efectuar una llamada al programa Java desde una estación de trabajo. Puede encontrar más información en la clase RunJavaApplication.

Clases JDBC

Los componentes de la interfaz gráfica de usuario de JDBC permiten a un programa Java presentar diversas vistas y controles para acceder a una base de datos utilizando sentencias y consultas SQL (lenguaje de consulta estructurada).

Los componentes disponibles son los siguientes:

- SQLStatementButton y SQLStatementMenuItem son, respectivamente, un botón o un elemento de menú que emite una sentencia SQL cuando se pulsa o selecciona.
- SQLStatementDocument: es un documento que se puede utilizar con cualquier componente gráfico de texto JFC (clases Java fundamentales) para emitir una sentencia SQL.
- SQLResultSetFormPane: es una sección que presenta los resultados de una consulta SQL en un formulario.
- SQLResultSetTablePane: es una sección que presenta los resultados de una consulta SQL en una tabla.
- SQLResultSetTableModel: es un modelo que gestiona los resultados de una consulta SQL en una tabla.
- SQLQueryBuilderPane: es una sección que presenta una herramienta interactiva que permite construir dinámicamente consultas SQL.

Todos los componentes de la interfaz gráfica de usuario de JDBC se comunican con la base de datos mediante un controlador JDBC. Este controlador debe registrarse en el gestor de controladores JDBC para hacer posible que funcionen todos estos componentes. El ejemplo siguiente registra el controlador JDBC de AS/400 Toolbox para Java:

```
        // Registre el controlador JDBC.
DriverManager.registerDriver (new com.ibm.as400.access.AS400JDBCDriver ());
```

Conexiones SQL

Un objeto SQLConnection representa una conexión con una base de datos mediante JDBC. **El objeto SQLConnection se utiliza con todos los componentes de la interfaz gráfica de usuario de JDBC.**

Para utilizar una conexión SQLConnection, establezca la propiedad URL mediante el constructor o con setURL(). Ello identifica la base de datos con la que se establece la conexión. Es posible establecer otras propiedades opcionales:

- Utilice setProperties() para especificar un conjunto de propiedades de conexión de JDBC.
- Utilice setUserName() para especificar el nombre de usuario correspondiente a la conexión.
- Utilice setPassword() para especificar la contraseña correspondiente a la conexión.

La conexión real con la base de datos no se realiza en el momento de crear el objeto SQLConnection. En vez de ello, dicha conexión se realiza al llamar al método getConnection(). Lo normal es que los componentes de la interfaz gráfica de usuario de JDBC llamen a este método de manera automática, pero también es posible llamarlo en cualquier momento para controlar cuándo se realiza la conexión.

En el ejemplo siguiente, se crea e inicializa un objeto SQLConnection:

```
        // Cree un objeto SQLConnection.
    SQLConnection connection = new SQLConnection ();

        // Establezca las propiedades URL y nombre de usuario de la
conexión.
    connection.setURL ("jdbc:as400://MySystem");
    connection.setUserName ("Lisa");
```

Se puede utilizar un objeto SQLConnection para más de un componente de la interfaz gráfica de usuario de JDBC.

Todos esos componentes emplearán la misma conexión, lo cual puede aumentar el rendimiento y la utilización de recursos. Alternativamente, cada componente de la interfaz gráfica de usuario de JDBC puede usar un objeto SQL distinto. A veces es necesario usar conexiones distintas, para que las sentencias SQL se emitan en distintas transacciones.

Cuando la conexión deje de ser necesaria, cierre el objeto SQLConnection mediante el método close(). Así se liberan los recursos de JDBC, tanto en el cliente como en el servidor.

Botones y elementos de menú

Un objeto SQLStatementButton representa un botón que, cuando se pulsa, emite una sentencia SQL (lenguaje de consulta estructurada). La clase SQLStatementButton amplía la clase JButton JFC (clases Java fundamentales) para que todos los botones tengan un aspecto y un comportamiento coherentes.

De forma parecida, un objeto SQLStatementMenuItem representa un elemento de menú que, cuando se selecciona, emite una sentencia SQL. La clase SQLStatementMenuItem amplía la clase JMenuItem JFC para que todos los elementos de menú tengan un aspecto y un comportamiento coherentes.

Para utilizar una de estas clases, establezca las propiedades connection y SQLStatement. Estas propiedades se pueden establecer mediante un constructor o con los métodos setConnection() y setSQLStatement().

A continuación hay un ejemplo en el que se crea un botón SQLStatementButton. El botón, cuando se pulsa en tiempo de ejecución, suprime todos los registros de una tabla:

```
    // Cree un objeto SQLStatementButton.
    // El texto del botón dice "Suprimir todo"
    // y no hay ningún icono.
SQLStatementButton button = new SQLStatementButton ("Suprimir todo");

    // Establezca las propiedades connection y
    // SQLStatement. Supongamos que "connection"
    // es un objeto SQLConnection que se ha
    // creado e inicializado en otra parte.
button.setConnection (connection);
button.setSQLStatement ("DELETE FROM MYTABLE");

    // Añada el botón a un marco. Supongamos
    // que "frame" es un objeto JFrame
    // creado en otra parte.
frame.getContentPane ().add (button);
```

Después de emitida la sentencia SQL, utilice el método getResultSet(), getMoreResults(), getUpdateCount() o getWarnings() para recuperar los resultados.

Clase SQLStatementDocument

La clase SQLStatementDocument es una implementación de la interfaz Document JFC (clases Java fundamentales). Esta clase se puede utilizar directamente con cualquier componente gráfico de texto JFC. En JFC hay varios componentes de texto, como pueden ser los campos de una sola línea (JTextField) y las áreas de texto de varias líneas (JTextArea). Los objetos SQLStatementDocument hacen que el contenido de los componentes de texto se asocie a los objetos SQLConnection. El programa Java puede ejecutar la sentencia SQL que se encuentra en el contenido del documento, en cualquier momento, y luego procesar los resultados, de haberlos.

Para utilizar un objeto SQLStatementDocument, debe establecer la propiedad connection. Para ello, utilice el constructor o el método setConnection(). El objeto SQLStatementDocument se "conecta" entonces al componente de texto, utilizando habitualmente el constructor o el método setDocument() del componente de texto. Utilice execute() en cualquier momento para ejecutar la sentencia SQL contenida en el documento.

A continuación hay un ejemplo en el que se crea un documento SQLStatementDocument en un campo JTextField:

```
        // Cree un objeto SQLStatementDocument.
        // Supongamos que "connection" es
        // un objeto SQLConnection que se ha
        // creado e inicializado en otra parte.
        // El texto del documento se
        // inicializa en una consulta genérica.
    SQLStatementDocument document = new SQLStatementDocument (connection,
"SELECT * FROM QIWS.QCUSTCDT");

        // Cree un campo de texto para presentar
        // el documento.
    JTextField textField = new JTextField ();
    textField.setDocument (document);

        // Añada el campo de texto a un marco.
   // Supongamos que "frame" es un objeto JFrame
   // creado en otra parte.
    frame.getContentPane ().add (textField);

        // Ejecute la sentencia SQL que está en
        // el campo de texto.
    document.execute ();
```

Después de emitida la sentencia SQL, utilice el método getResultSet(), getMoreResults(), getUpdateCount() o getWarnings() para recuperar los resultados.

Clase SQLResultSetFormPane

Una sección SQLResultSetFormPane presenta los resultados de una consulta SQL (lenguaje de consulta estructurada) en un formulario. El formulario visualiza los registros de uno en uno y proporciona botones que permiten al usuario desplazarse hacia delante, hacia atrás, al primer o al último registro, o renovar la vista de los resultados.

Para utilizar una sección SQLResultSetFormPane, establezca las propiedades connection y query. Estas propiedades se establecen mediante el constructor o con los métodos setConnection() y setQuery(). Utilice load() para ejecutar la consulta y presentar el primer registro del conjunto de resultados. Cuando ya no necesite los resultados, llame a close() para asegurarse de que se cierre el conjunto de resultados.

El siguiente ejemplo crea un objeto SQLResultSetFormPane y lo añade a un marco:

```
         // Cree un objeto SQLResultSetFormPane.
         // Supongamos que "connection" es
         // un objeto SQLConnection que se ha
         // creado e inicializado en otra parte.
     SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection,
"SELECT * FROM QIWS.QCUSTCDT");

         // Cargue los resultados.
     formPane.load ();

         // Añada la sección formulario a un marco.
    // Supongamos que "frame" es un objeto JFrame
    // creado en otra parte.
     frame.getContentPane ().add (formPane);
```

Clase SQLResultSetTablePane

Una sección SQLResultSetTablePane es un objeto que presenta los resultados de una consulta SQL (lenguaje de consulta estructurada) en una tabla. Cada fila de la tabla visualiza un registro del conjunto de resultados y cada columna visualiza un campo.

Para utilizar una sección SQLResultSetTablePane, establezca las propiedades connection y query. Establezca las propiedades mediante el constructor o con los métodos setConnection() y setQuery(). Utilice load() para ejecutar la consulta y presentar los resultados en la tabla. Cuando ya no necesite los resultados, llame a close() para asegurarse de que se cierre el conjunto de resultados.

El siguiente ejemplo crea un objeto SQLResultSetTablePane y lo añade a un marco:

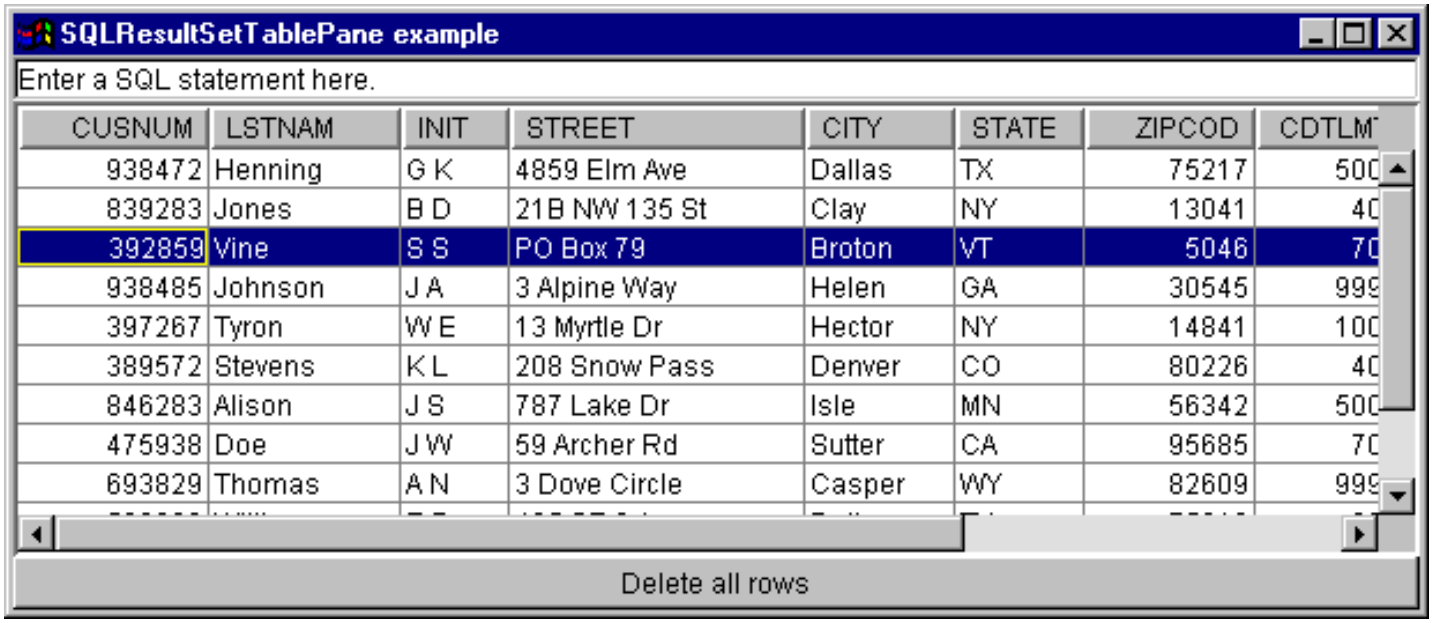
```
        // Cree un objeto SQLResultSetTablePane.
        // Suponga que "connection" es
        // un objeto SQLConnection que se ha
        // creado e inicializado en otra parte.
    SQLResultSetTablePane tablePane = new SQLResultSetTablePane
(connection, "SELECT * FROM QIWS.QCUSTCDT");

        // Cargue los resultados.
    tablePane.load ();

        // Añada la sección tabla a un marco.
        // Suponga que "frame" es un objeto JFrame
        // creado en otra parte.
    frame.getContentPane ().add (tablePane);
```

Ejemplo

Presentar una sección SQLResultSetTablePane que visualiza el contenido de una tabla. Este ejemplo utiliza un documento SQLStatementDocument (indicado en la siguiente imagen mediante el texto "Escriba aquí una sentencia SQL") que permite al usuario escribir cualquier sentencia SQL, y utiliza además un botón SQLStatementButton (indicado mediante el texto "Suprimir todas las filas") que permite al usuario suprimir todas las filas de la tabla.

La figura 1 muestra el componente de la interfaz gráfica de usuario SQLResultSetTablePane:

Figura 1: componente GUI SQLResultSetTablePane



Clase SQLResultSetTableModel

La sección SQLResultSetTablePane se implementa mediante el paradigma controlador de modelos-vistas, en el que los datos y la interfaz de usuario están separados en distintas clases. La implementación integra SQLResultSetTableModel en la clase JTable de JFC (clases Java fundamentales). La clase SQLResultSetTableModel gestiona los resultados de la consulta, y la clase JTable visualiza los resultados gráficamente y maneja la interacción de usuario.

SQLResultSetTablePane proporciona suficientes funciones para la mayoría de las necesidades. No obstante, un llamador, si necesita un mayor control del componente JFC, puede utilizar la clase SQLResultSetTableModel directamente y proporcionar una integración personalizada en otro componente de la interfaz gráfica de usuario.

Para utilizar un modelo SQLResultSetTableModel, establezca las propiedades connection y query. Estas propiedades se establecen mediante el constructor o con los métodos setConnection() y setQuery(). Utilice load() para ejecutar la consulta y cargar los resultados. Cuando ya no necesite los resultados, llame a close() para asegurarse de que se cierre el conjunto de resultados.

El siguiente ejemplo crea un objeto SQLResultSetTableModel y lo presenta en una tabla JTable:

```
        // Cree un objeto SQLResultSetTableModel.
        // Supongamos que "connection" es
        // un objeto SQLConnection que se ha
        // creado e inicializado en otra parte.
    SQLResultSetTableModel tableModel = new SQLResultSetTableModel
(connection, "SELECT * FROM QIWS.QCUSTCDT");

        // Cargue los resultados.
    tableModel.load ();

        // Cree una tabla JTable para el modelo.
    JTable table = new JTable (tableModel);

        // Añada la tabla a un marco. Supongamos
        // que "frame" es un objeto JFrame
        // creado en otra parte.
    frame.getContentPane ().add (table);
```

Constructores de consultas SQL

Una sección SQLQueryBuilderPane presenta una herramienta interactiva que permite construir dinámicamente consultas SQL.

Para utilizar un objeto SQLQueryPane, debe establecer la propiedad connection. Esta propiedad se puede establecer mediante el constructor o con el método setConnection(). Utilice load() para cargar los datos necesarios para la interfaz gráfica de usuario del constructor de consultas. Utilice getQuery() para obtener la consulta SQL construida por el usuario.

El siguiente ejemplo crea un objeto SQLQueryBuilderPane y lo añade a un marco:

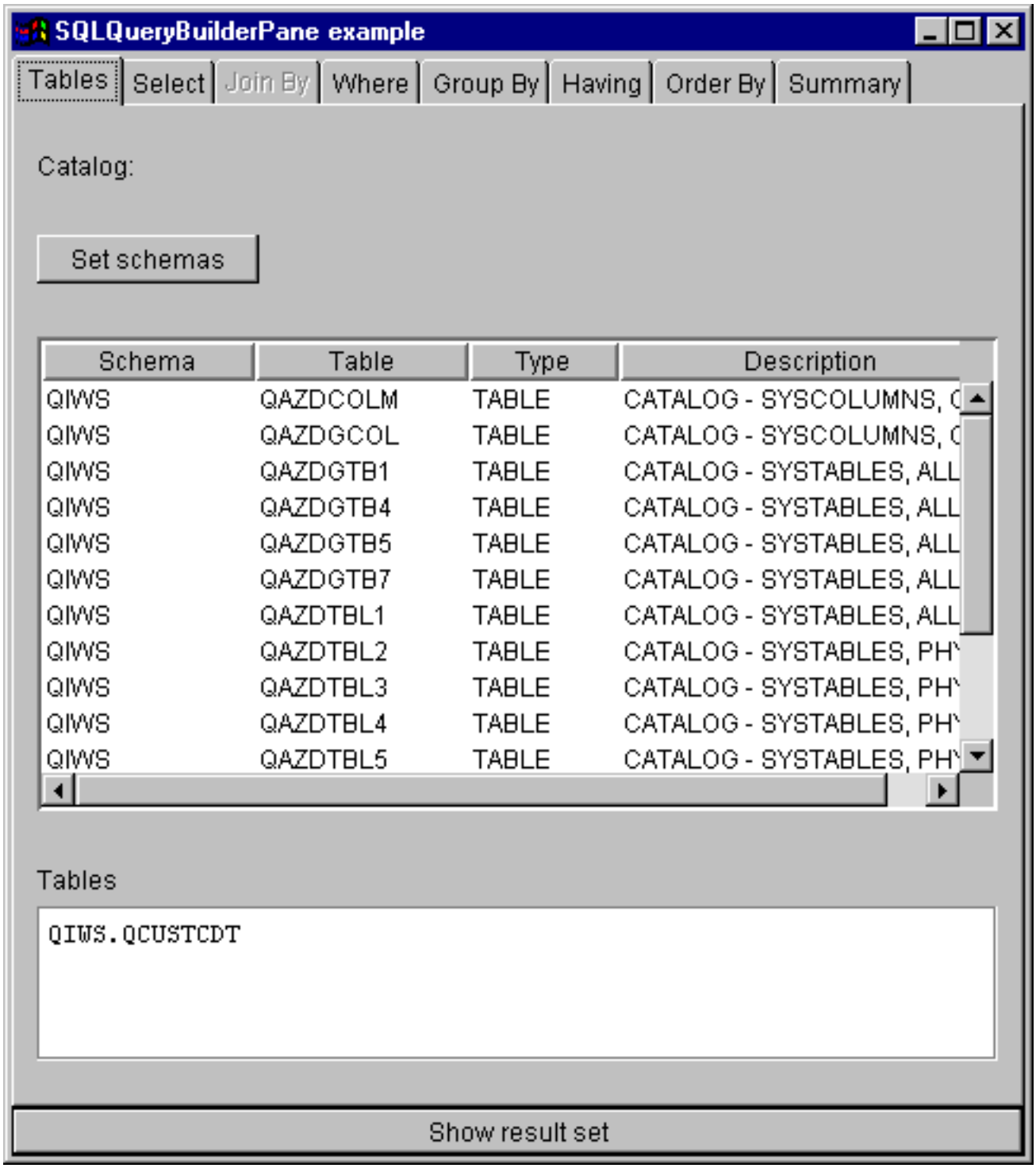
```
        // Cree un objeto SQLQueryBuilderPane.
        // Supongamos que "connection" es
        // un objeto SQLConnection que se ha
        // creado e inicializado en otra parte.
    SQLQueryBuilderPane queryBuilder = new SQLQueryBuilderPane
(connection);

        // Cargue los datos necesarios para el
        // constructor de consultas.
    queryBuilder.load ();

        // Añada la sección del panel constructor de consultas a un
        // marco. Supongamos que "frame" es un objeto JFrame
        // creado en otra parte.
    frame.getContentPane ().add (queryBuilder);
```

Ejemplo

Presentar una sección SQLQueryBuilderPane y un botón. Al pulsar el botón, presentar los resultados de la consulta en una sección SQLResultSetFormPane, en otro marco.

La figura 1 muestra el componente de la interfaz gráfica de usuario SQLQueryBuilderPane:

Figura 1: componente GUI SQLQueryBuilderPane


SQLQueryBuilderPane example
Tables
Select
Join By
Where
Group By
Having
Order By
Summary
Catalog:
Set schemas
Schema
Table
Type
Description
QIWS QAZDCOLM TABLE CATALOG - SYSCOLUMNS, C
QIWS QAZDGCOL TABLE CATALOG - SYSCOLUMNS, C
QIWS QAZDGTB1 TABLE CATALOG - SYSTABLES, ALL
QIWS QAZDGTB4 TABLE CATALOG - SYSTABLES, ALL
QIWS QAZDGTB5 TABLE CATALOG - SYSTABLES, ALL
QIWS QAZDGTB7 TABLE CATALOG - SYSTABLES, ALL
QIWS QAZDTBL1 TABLE CATALOG - SYSTABLES, ALL
QIWS QAZDTBL2 TABLE CATALOG - SYSTABLES, PH
QIWS QAZDTBL3 TABLE CATALOG - SYSTABLES, PH
QIWS QAZDTBL4 TABLE CATALOG - SYSTABLES, PH
QIWS QAZDTBL5 TABLE CATALOG - SYSTABLES, PH
Tables
QIWS.QCUSTCDT
Show result set

Trabajos

Los componentes de vaccess (GUI) de trabajos permiten a un programa Java presentar listas de trabajos y mensajes de anotaciones de trabajo de servidor en una interfaz gráfica de usuario.

Los componentes disponibles son los siguientes:

- Un objeto VJobList es un recurso que representa una lista de trabajos servidores para utilizarse en los objetos AS400Pane.
- Un objeto VJob es un recurso que representa la lista de mensajes de unas anotaciones de trabajo para utilizarse en objetos AS400Pane.

Los objetos AS400Pane, VJobList y VJob se pueden utilizar conjuntamente para presentar diversas vistas de una lista de trabajos o de unas anotaciones de trabajo.

Para utilizar un objeto VJobList, establezca las propiedades system, name, number y user. Estas propiedades se establecen mediante un constructor o con los métodos setSystem(), setName(), setNumber() y setUser().

Para utilizar un objeto VJob, establezca la propiedad system. Esta propiedad se establece mediante un constructor o con el método setSystem().

Luego, el objeto VJobList o VJob se "conecta" al objeto AS400Pane como raíz; para ello se utiliza el constructor de la sección o el método setRoot().

VJobList dispone de otras propiedades que son de utilidad para definir el conjunto de trabajos presentados en los objetos AS400Pane. Utilice setName() para especificar que únicamente deben mostrarse los trabajos que tengan un nombre determinado. Utilice setNumber() para especificar que únicamente deben mostrarse los trabajos que tengan un número determinado. De forma parecida, utilice setUser() para especificar que únicamente deben mostrarse los trabajos correspondientes a un usuario concreto.

Los objetos AS400Pane, VJobList y VJob, cuando se crean, se inicializan en un estado por omisión. La lista de trabajos o los mensajes de las anotaciones de trabajo no se cargan en el momento de la creación de tales objetos. Para cargar el contenido, el llamador debe llamar explícitamente al método load() en cualquiera de los objetos. Ello hará que se inicie la comunicación con el servidor para recopilar el contenido de la lista.

En tiempo de ejecución, pulse el botón derecho del ratón en un trabajo, en una lista de trabajos o en un mensaje de anotaciones de trabajo para visualizar el menú de atajo. Seleccione **Propiedades** en el menú de atajo para llevar a cabo acciones en el objeto seleccionado:

- Trabajo: permite trabajar con las propiedades, como por ejemplo el tipo y el estado. También puede cambiar el valor de algunas de las propiedades.
- Lista de trabajos: permite trabajar con las propiedades, como por ejemplo el nombre, el número y el usuario. También puede cambiar el contenido de la lista.
- Mensaje de anotaciones de trabajo: permite visualizar las propiedades, como por ejemplo el texto completo, la gravedad y la hora de envío.

Los usuarios únicamente pueden acceder a los trabajos sobre los que poseen autorización. Además, el programa Java puede impedir que el usuario realice determinadas acciones mediante el método setAllowActions() en la sección.

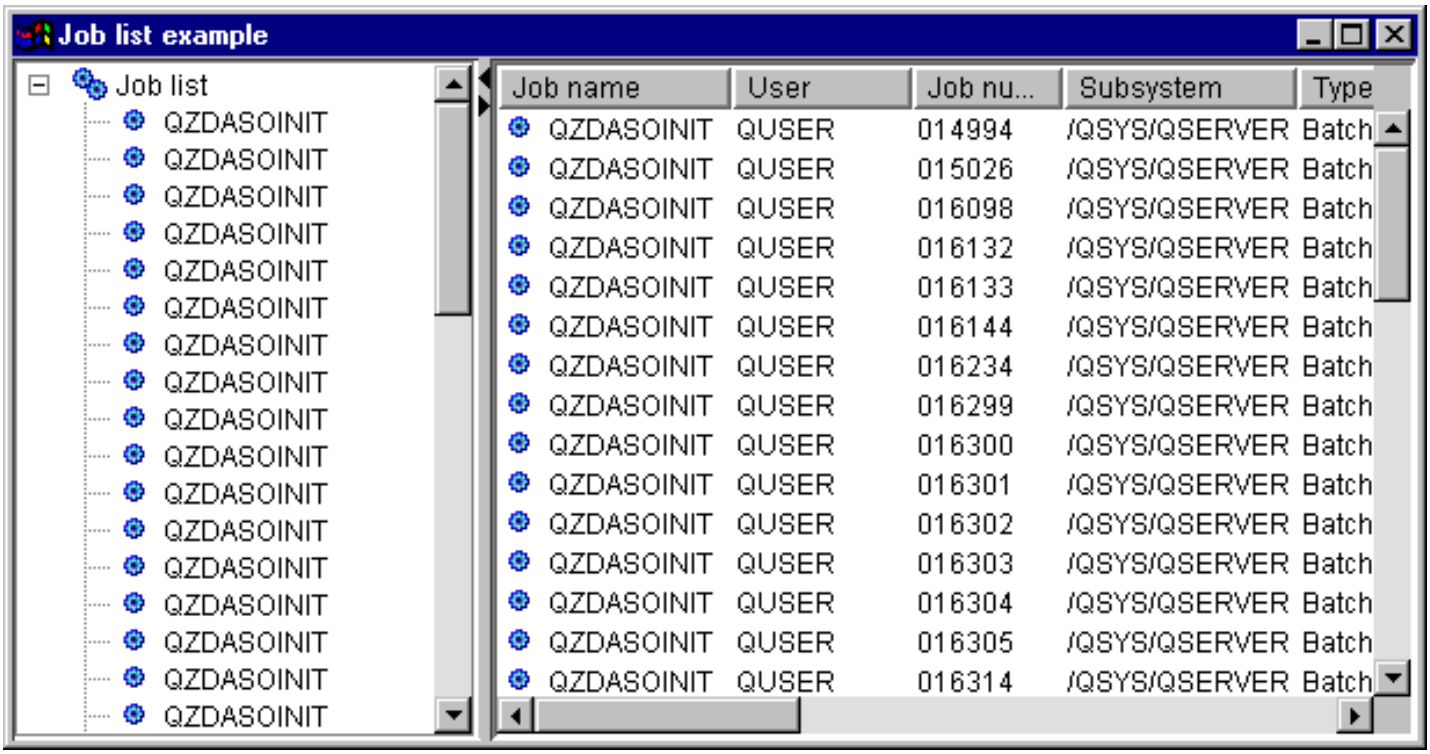
En el ejemplo siguiente se crea una lista VJobList y se presenta en una sección AS400ExplorerPane:

```
        // Cree el objeto VJobList. Supongamos
        // que "system" es un objeto AS400
        // creado e inicializado en otra parte.
 VJobList root = new VJobList (system);

// Cree y cargue un objeto
     // AS400ExplorerPane.
 AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
                explorerPane.load ();

     // Añada la sección de panel explorador a un marco.
// Supongamos que "frame" es un objeto JFrame
// creado en otra parte.
 frame.getContentPane ().add (explorerPane);
```

Ejemplos

Este ejemplo de VJobList presenta una sección AS400ExplorerPane que se rellena con una lista de trabajos. La lista muestra los trabajos que tienen un nombre idéntico en el sistema.

La imagen siguiente muestra el componente de la interfaz gráfica de usuario de VJobList:



Clases de mensajes de vaccess

Los componentes de la interfaz gráfica de usuario de mensajes permiten a un programa Java presentar listas de mensajes de servidor en una interfaz gráfica de usuario.

Los componentes disponibles son los siguientes:

- Un objeto Lista de mensajes: es un recurso que representa una lista de mensajes para utilizarse en los objetos AS400Pane. Sirve para las listas de mensajes generadas por las llamadas a mandato o a programa.
- Un objeto Colas de mensajes: es un recurso que representa los mensajes que hay en una cola de mensajes de servidor para utilizarse en los objetos AS400Pane.

Los objetos AS400Pane son componentes de la interfaz gráfica de usuario que presentan uno o varios recursos de servidor y permiten su manipulación. Los objetos VMessageList y VMessageQueue son recursos que representan listas de mensajes de servidor en los objetos AS400Pane.

Los objetos AS400Pane, VMessageList y VMessageQueue se pueden utilizar conjuntamente para presentar múltiples vistas de una lista de mensajes y permitir al usuario seleccionar y realizar operaciones en los mensajes.

Clase VMessageList

Un objeto VMessageList es un recurso que representa una lista de mensajes para utilizarse en los objetos AS400Pane. Sirve para las listas de mensajes generadas por las llamadas a mandato o a programa. Los métodos que devuelven listas de mensajes son los siguientes:

- CommandCall.getMessageList()
- CommandCallButton.getMessageList()
- CommandCallMenuItem.getMessageList()
- ProgramCall.getMessageList()
- ProgramCallButton.getMessageList()
- ProgramCallMenuItem.getMessageList()

Para utilizar una lista VMessageList, establezca la propiedad messageList. Esta propiedad se establece mediante un constructor o con el método setMessageList(). Posteriormente, el objeto VMessageList se "conecta" al objeto AS400Pane como raíz, utilizando el constructor o el método setRoot() del objeto AS400Pane.

Los objetos AS400Pane y VMessageList, cuando se crean, se inicializan en un estado por omisión. La lista de mensajes no se carga en el momento de la creación de tales objetos. Para cargar el contenido, el llamador ha de llamar explícitamente al método load() en cualquiera de los objetos.

En tiempo de ejecución, un usuario puede llevar a cabo acciones en un mensaje pulsando sobre él con el botón derecho del ratón para visualizar el menú de contexto. El menú de contexto de los mensajes puede contener un elemento denominado **Propiedades** que visualiza propiedades tales como la gravedad, el tipo y la fecha.

El llamador puede impedir que el usuario lleve a cabo acciones, utilizando para ello el método setAllowActions() en la sección.

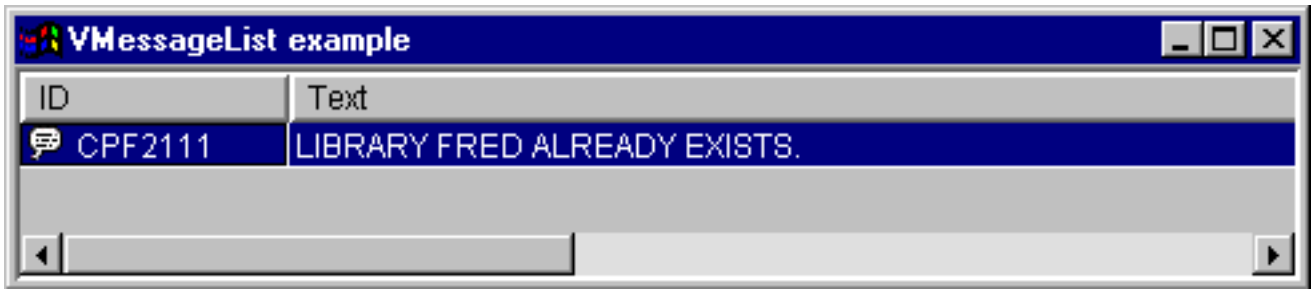
En el ejemplo siguiente se crea una lista VMessageList para los mensajes generados por una llamada a mandato y se presenta dicha lista en una sección AS400DetailsPane:

```
        // Cree el objeto VMessageList.
        // Suponga que "command" es un
        // objeto CommandCall creado y ejecutado
        // en otra parte.
    VMessageList root = new VMessageList (command.getMessageList ());

        // Cree y cargue un objeto
        // AS400DetailsPane.
    AS400DetailsPane detailsPane = new AS400DetailsPane (root);
           detailsPane.load ();

   // Añada la sección de detalles a un marco.
   // Suponga que "frame" es un objeto JFrame
   // creado en otra parte.
    frame.getContentPane ().add (detailsPane);
```

Ejemplo

Presentar la lista de mensajes generados por una llamada a mandato utilizando un objeto AS400DetailsPane con un objeto VMessageList. La figura 1 muestra el componente de la interfaz gráfica de usuario VMessageList:

Figura 1: componente GUI VMessageList


VMessageList example
ID
Text
CPF2111
LIBRARY FRED ALREADY EXISTS.

Clase VMessageQueue

Un objeto VMessageQueue es un recurso que representa los mensajes de una cola de mensajes de servidor para utilizarse en los objetos AS400Pane.

Para utilizar una cola VMessageQueue, establezca las propiedades system y path. Estas propiedades se pueden establecer mediante un constructor o con los métodos setSystem() y setPath(). Posteriormente, el objeto VMessageQueue se "conecta" al objeto AS400Pane como raíz utilizando el constructor o el método setRoot() del objeto AS400Pane.

VMessageQueue dispone de otras propiedades que son de utilidad para definir el conjunto de mensajes presentados en los objetos AS400Pane. Utilice setSeverity() para especificar la gravedad de los mensajes que deben aparecer. Utilice setSelection() para especificar el tipo de los mensajes que deben aparecer.

Los objetos AS400Pane y VMessageQueue, cuando se crean, se inicializan en un estado por omisión. La lista de mensajes no se carga en el momento de la creación de tales objetos. Para cargar el contenido, el llamador ha de llamar explícitamente al método load() en cualquiera de los objetos. Ello hará que se inicie la comunicación con el servidor para recopilar el contenido de la lista.

En tiempo de ejecución, un usuario puede llevar a cabo acciones en un mensaje o una cola de mensajes pulsando sobre ese objeto con el botón derecho del ratón para visualizar el menú de contexto. El menú de contexto de las colas de mensajes puede contener los elementos siguientes:

- **Borrar** - Borra la cola de mensajes.
- **Propiedades** - Permite al usuario establecer las propiedades de gravedad y selección. Esta acción se puede utilizar para cambiar el contenido de la lista.

Para los mensajes de una cola de mensajes, se dispone de estas acciones:

- **Eliminar** - Elimina el mensaje de la cola de mensajes.
- **Responder** - Responde a un mensaje de consulta.
- **Propiedades** - Visualiza propiedades tales como la gravedad, el tipo y la fecha.

Como es natural, los usuarios únicamente pueden acceder a las colas de mensajes sobre las que poseen autorización. Además, el llamador puede impedir que el usuario lleve a cabo acciones utilizando para ello el método setAllowActions() en la sección.

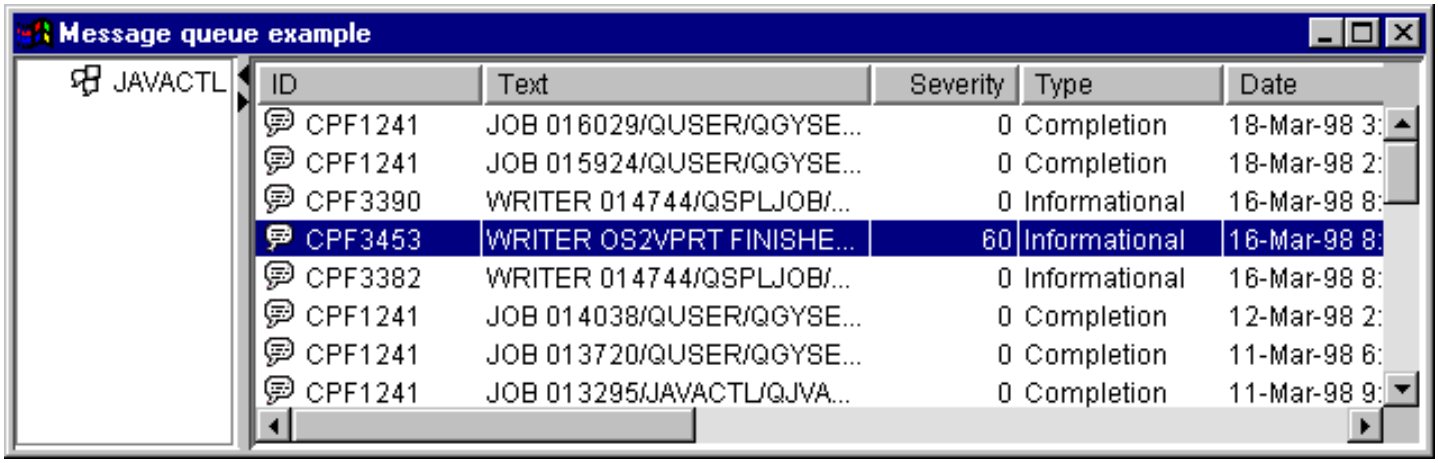
En el ejemplo siguiente se crea una cola VMessageQueue y se presenta en una sección AS400ExplorerPane:

```
         // Cree el objeto VMessageQueue.
         // Suponga que "system" es un objeto
         // AS400 creado e inicializado
         // en otra parte.
     VMessageQueue root = new VMessageQueue (system,
"/QSYS.LIB/MYLIB.LIB/MYMSGQ.MSGQ");

    // Cree y cargue un objeto
         // AS400ExplorerPane.
     AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
                   explorerPane.load ();

         // Añada la sección de panel explorador a un marco.
    // Suponga que "frame" es un objeto JFrame
    // creado en otra parte.
     frame.getContentPane ().add (explorerPane);
```

Ejemplo

Presentar la lista de mensajes de una cola de mensajes utilizando un objeto AS400ExplorerPane con un objeto VMessageQueue. La figura 1 muestra el componente de la interfaz gráfica de usuario VMessageQueue:

Figura 1: componente GUI VMessageQueue


Message queue example
JAVACTL
ID
Text
Severity
Type
Date
CPF1241 JOB 016029/QUSER/QGYSE... 0 Completion 18-Mar-98 3:
CPF1241 JOB 015924/QUSER/QGYSE... 0 Completion 18-Mar-98 2:
CPF3390 WRITER 014744/QSPLJOB/... 0 Informational 16-Mar-98 8:
CPF3453 WRITER OS2VPRT FINISHE... 60 Informational 16-Mar-98 8:
CPF3382 WRITER 014744/QSPLJOB/... 0 Informational 16-Mar-98 8:
CPF1241 JOB 014038/QUSER/QGYSE... 0 Completion 12-Mar-98 2:
CPF1241 JOB 013720/QUSER/QGYSE... 0 Completion 11-Mar-98 6:
CPF1241 JOB 013295/JAVACTL/QJVA... 0 Completion 11-Mar-98 9:

Clases de permisos

Se puede utilizar la información de las clases de permiso en una interfaz gráfica de usuario (GUI) mediante las clases VIFSFile y VIFSDirectory. El permiso se ha añadido en forma de acción en cada una de estas clases.

El siguiente ejemplo muestra cómo se utiliza la clase Permission con la clase VIFSDirectory:

```
// Cree un objeto AS400.
AS400 as400 = new AS400();

// Cree un directorio IFSDirectory utilizando el nombre del sistema
// y la vía de acceso completa de un objeto QSYS.
VIFSDirectory directory = new VIFSDirectory(as400,
                              "/QSYS.LID/testlib1.lib");

// Cree una sección de panel explorador.
AS400ExplorerPane pane = new AS400ExplorerPane((VNode)directory);

// Cargue la información.
pane.load();
```

Clases de impresión de vaccess

Los siguientes componentes del paquete vaccess permiten a un programa Java presentar listas de recursos de impresión de servidor en una interfaz gráfica de usuario:

- Un objeto VPrinters es un recurso que representa una lista de impresoras para utilizarse en los objetos AS400Pane.
- Un objeto VPrinter es un recurso que representa una impresora y sus archivos en spool para utilizarse en los objetos AS400Pane.
- Un objeto VPrinterOutput es un recurso que representa una lista de archivos en spool para utilizarse en los objetos AS400Pane.
- Un objeto SpooledFileViewer es un recurso que representa visualmente los archivos en spool.

Las secciones de panel AS/400 (AS400Pane) son componentes de la interfaz gráfica de usuario que presentan uno o varios recursos del servidor y permiten su manipulación. Los objetos VPrinters, VPrinter y VPrinterOutput son recursos que representan listas de recursos de impresión de servidor en objetos AS400Pane.

Los objetos AS400Pane, VPrinters, VPrinter y VPrinterOutput se pueden utilizar conjuntamente para presentar múltiples vistas de recursos de impresión y permitir al usuario seleccionarlos y efectuar operaciones en ellos.

Clase VPrinters

Un objeto VPrinters es un recurso que representa una lista de impresoras para utilizarse en los objetos AS400Pane.

Para utilizar un objeto VPrinters, establezca la propiedad system. Esta propiedad se establece mediante un constructor o con el método setSystem(). Posteriormente, el objeto VPrinters se "conecta" al objeto AS400Pane como raíz utilizando el constructor o el método setRoot() de la sección.

Un objeto VPrinters dispone de otra propiedad muy útil para definir el conjunto de impresoras que se presenta en los objetos AS400Pane. Utilice setPrinterFilter() para especificar un filtro que define qué impresoras deben aparecer.

Los objetos AS400Pane y VPrinters, cuando se crean, se inicializan en un estado por omisión. La lista de impresoras no se ha cargado. Para cargar el contenido, el llamador ha de llamar explícitamente al método load() en cualquiera de los objetos.

En tiempo de ejecución, un usuario puede llevar a cabo acciones en cualquier lista de impresoras o impresora pulsando sobre ella con el botón derecho del ratón para visualizar el menú de contexto. El menú de contexto de las listas de impresoras puede contener un elemento denominado **Propiedades** que permite al usuario establecer la propiedad de filtro de impresoras, que puede cambiar el contenido de la lista.

El menú de contexto de las impresoras puede contener los elementos siguientes:

- **Retener** - Retiene la impresora.
- **Liberar** - Libera la impresora.
- **Iniciar** - Inicia la impresora.
- **Detener** - Detiene la impresora.
- **Hacer disponible** - Hace que la impresora esté disponible.
- **Hacer no disponible** - Hace que la impresora no esté disponible.
- **Propiedades** - Visualiza las propiedades de la impresora y permite al usuario establecer filtros.

Los usuarios únicamente pueden acceder a las impresoras sobre las que poseen autorización. Además, el llamador puede impedir que el usuario lleve a cabo acciones utilizando para ello el método setAllowActions() en la sección.

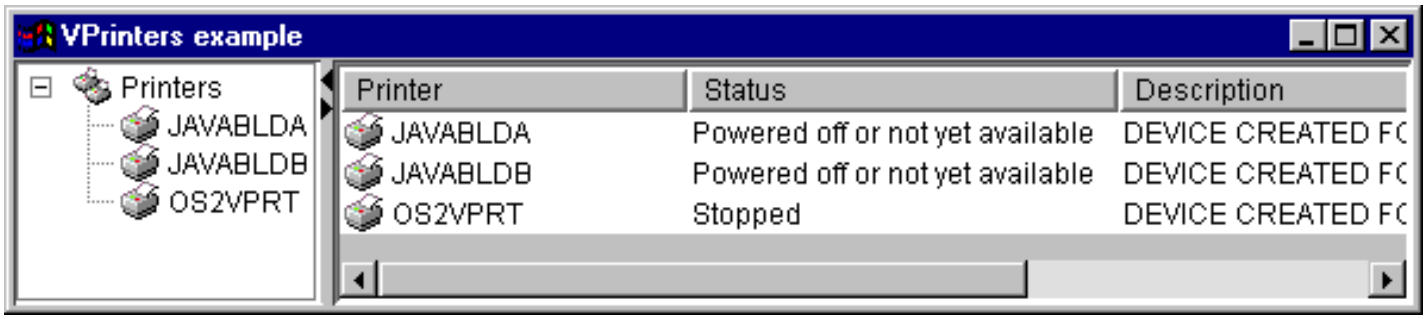
En el ejemplo siguiente se crea un objeto VPrinters y se presenta en una sección AS400TreePane:

```
        // Cree el objeto VPrinters.
        // Suponga que "system" es un objeto
        // AS400 creado e inicializado
        // en otra parte.
    VPrinters root = new VPrinters (system);

        // Cree y cargue un objeto
        // AS400TreePane.
    AS400TreePane treePane = new AS400TreePane (root);
    treePane.load ();

// Añada la sección en árbol a un marco.
// Suponga que "frame" es un objeto JFrame
// creado en otra parte.
    frame.getContentPane ().add (treePane);
```

Ejemplo

Presentar recursos de impresión utilizando una sección AS400ExplorerPane con un objeto VPrinters. La figura 1 muestra el componente de la interfaz gráfica de usuario VPrinters:

Figura 1: componente GUI VPrinters


VPrinters example
Printers
JAVABLDA
JAVABLDB
OS2VPRT
Printer
Status
Description
JAVABLDA
Powered off or not yet available
DEVICE CREATED FC
JAVABLDB
Powered off or not yet available
DEVICE CREATED FC
OS2VPRT
Stopped
DEVICE CREATED FC

VPrinter

Un objeto VPrinter es un recurso que representa una impresora de servidor y sus archivos en spool para utilizarse en los objetos AS400Pane.

Para utilizar un objeto VPrinter, establezca la propiedad printer. Esta propiedad se establece mediante un constructor o con el método setPrinter(). Posteriormente, el objeto VPrinter se "conecta" al objeto AS400Pane como raíz utilizando el constructor o el método setRoot() de la sección.

Los objetos AS400Pane y VPrinter, cuando se crean, se inicializan en un estado por omisión. Los atributos de la impresora y la lista de archivos en spool no se cargan en el momento de la creación.

Para cargar el contenido, el llamador debe llamar explícitamente al método load() en cualquiera de los objetos. Ello hará que se inicie la comunicación con el servidor para recopilar el contenido de la lista.

En tiempo de ejecución, un usuario puede llevar a cabo acciones en cualquier impresora o archivo en spool pulsando sobre ese objeto con el botón derecho del ratón para visualizar el menú de contexto. El menú de contexto de las colas de mensajes puede contener los elementos siguientes:

- **Retener** - Retiene la impresora.
- **Liberar** - Libera la impresora.
- **Iniciar** - Inicia la impresora.
- **Detener** - Detiene la impresora.
- **Hacer disponible** - Hace que la impresora esté disponible.
- **Hacer no disponible** - Hace que la impresora no esté disponible.
- **Propiedades** - Visualiza las propiedades de la impresora y permite al usuario establecer filtros.

El menú de contexto de los archivos en spool listados para una impresora puede contener los elementos siguientes:

- **Responder** - Responde al archivo en spool.
- **Retener** - Retiene el archivo en spool.
- **Liberar** - Libera el archivo en spool.
- **Imprimir siguiente** - Imprime el siguiente archivo en spool.
- **Enviar** - Envía el archivo en spool.
- **Mover** - Mueve el archivo en spool.
- **Suprimir** - Suprime el archivo en spool.
- **Propiedades** - Visualiza múltiples propiedades del archivo en spool y permite al usuario realizar cambios en algunas de ellas.

Los usuarios únicamente pueden acceder a las impresoras y a los archivos en spool sobre los que poseen autorización. Además, el llamador puede impedir que el usuario lleve a cabo acciones utilizando para ello el método setAllowActions() en la sección.

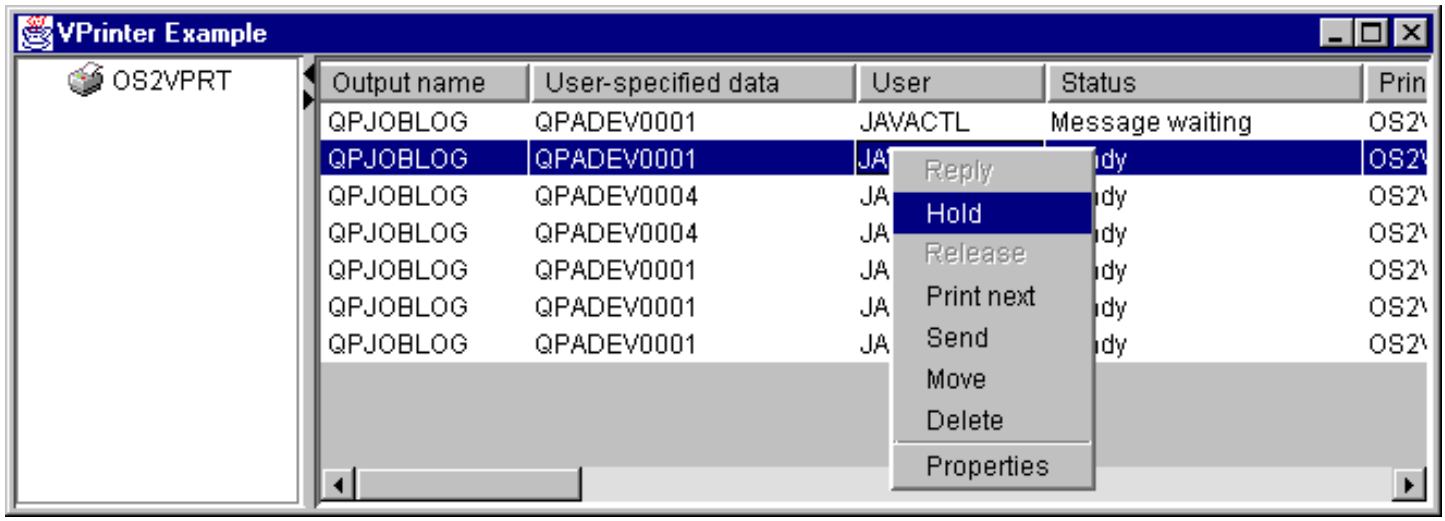
En el ejemplo siguiente se crea una impresora VPrinter y se presenta en una sección AS400ExplorerPane:

```
        // Cree el objeto VPrinter.
        // Suponga que "system" es un objeto
        // AS400 creado e inicializado
        // en otra parte.
 VPrinter root = new VPrinter (new Printer (system, "MYPRINTER"));

// Cree y cargue un objeto
     // AS400ExplorerPane.
 AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
                explorerPane.load ();

// Añada la sección de panel explorador a un marco.
// Suponga que "frame" es un objeto JFrame
// creado en otra parte.
 frame.getContentPane ().add (explorerPane);
```

Ejemplo

Presentar recursos de impresión utilizando una sección AS400ExplorerPane con un objeto VPrinter. La figura 1 muestra el componente de la interfaz gráfica de usuario VPrinter:

Figura 1: componente GUI VPrinter


VPrinter Example
OS2VPRT
Output name
User-specified data
User
Status
Prin
QPJOBLOG
QPADEV0001
JAVACTL
Message waiting
OS2\
QPJOBLOG
QPADEV0001
JA
dy
OS2\
QPJOBLOG
QPADEV0004
JA
dy
OS2\
QPJOBLOG
QPADEV0004
JA
dy
OS2\
QPJOBLOG
QPADEV0001
JA
dy
OS2\
QPJOBLOG
QPADEV0001
JA
dy
OS2\
QPJOBLOG
QPADEV0001
JA
dy
OS2\
Reply
Hold
Release
Print next
Send
Move
Delete
Properties

Clase VPrinterOutput

Un objeto VPrinterOutput es un recurso que representa una lista de archivos en spool existentes en un servidor para utilizarse en los objetos AS400Pane.

Para utilizar un objeto VPrinterOutput, establezca la propiedad system. Esta propiedad se puede establecer mediante un constructor o con el método setSystem(). Posteriormente, el objeto VPrinterOutput se "conecta" al objeto AS400Pane como raíz, utilizando el constructor o el método setRoot() del objeto AS400Pane.

Un objeto VPrinterOutput dispone de otras propiedades que son de utilidad para definir el conjunto de archivos en spool que se presenta en los objetos AS400Pane. Utilice setFormTypeFilter() para especificar qué tipos de formularios deben aparecer. Utilice setUserDataFilter() para especificar qué datos de usuario deben aparecer. Por último, utilice setUserFilter() para especificar qué archivos en spool deben aparecer.

Los objetos AS400Pane y VPrinterOutput, cuando se crean, se inicializan en un estado por omisión. La lista de archivos en spool no se carga en el momento de la creación. Para cargar el contenido, el llamador ha de llamar explícitamente al método load() en cualquiera de los objetos. Ello hará que se inicie la comunicación con el servidor para recopilar el contenido de la lista.

En tiempo de ejecución, un usuario puede llevar a cabo acciones en cualquier archivo en spool o lista de archivos en spool pulsando sobre ese objeto con el botón derecho del ratón para visualizar el menú de contexto. El menú de contexto de las listas de archivos en spool puede contener un elemento denominado **Propiedades** que permite al usuario establecer las propiedades de filtro, que pueden cambiar el contenido de la lista.

El menú de contexto de los archivos en spool puede contener los elementos siguientes:

- **Responder** - Responde al archivo en spool.
- **Retener** - Retiene el archivo en spool.
- **Liberar** - Libera el archivo en spool.
- **Imprimir siguiente** - Imprime el siguiente archivo en spool.
- **Enviar** - Envía el archivo en spool.
- **Mover** - Mueve el archivo en spool.
- **Suprimir** - Suprime el archivo en spool.
- **Propiedades** - Visualiza múltiples propiedades del archivo en spool y permite al usuario realizar cambios en algunas de ellas.

Como es natural, los usuarios únicamente pueden acceder a los archivos en spool sobre los que poseen autorización. Además, el llamador puede impedir que el usuario lleve a cabo acciones utilizando para ello el método setAllowActions() en la sección.

En el ejemplo siguiente se crea una salida VPrinterOutput y se presenta en una sección AS400ListPane:

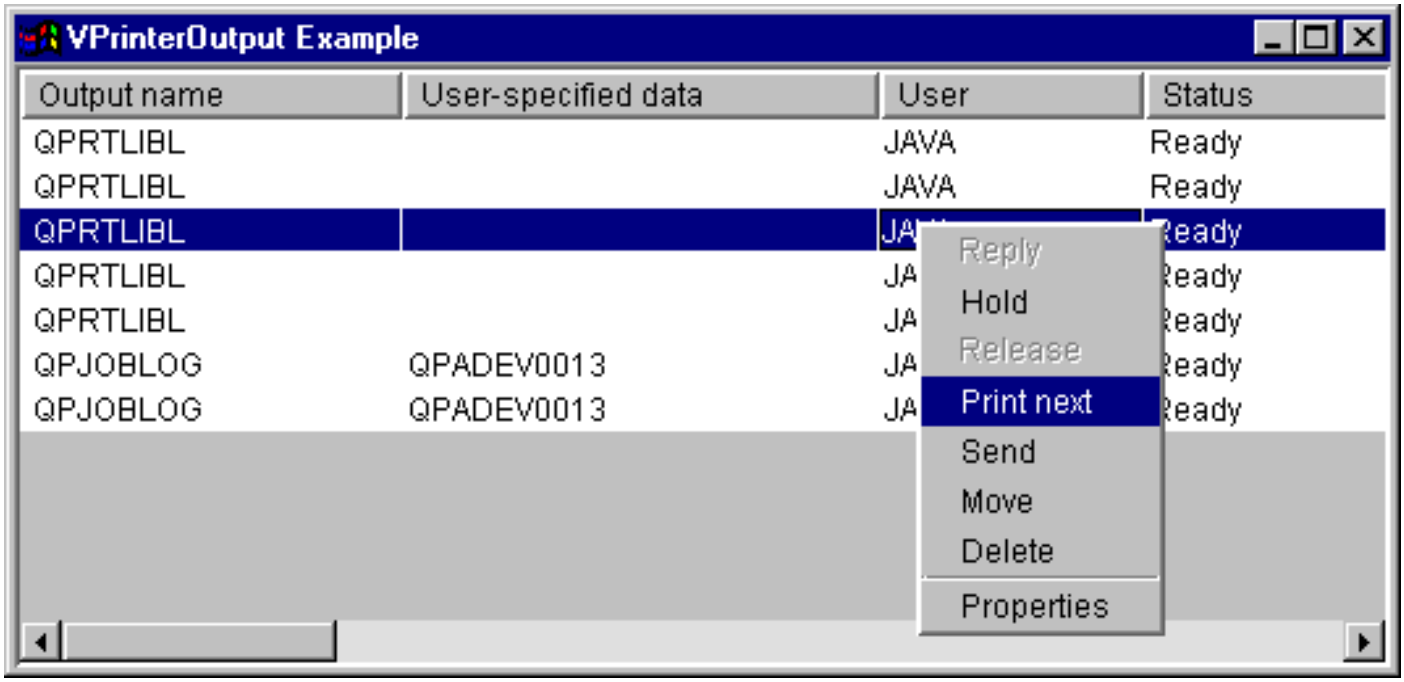
```
        // Cree el objeto VPrinterOutput.
        // Suponga que "system" es un objeto
        // AS400 creado e inicializado
        // en otra parte.
    VPrinterOutput root = new VPrinterOutput (system);

        // Cree y cargue un objeto
        // AS400ListPane.
    AS400ListPane listPane = new AS400ListPane (root);
    listPane.load ();

// Añada la sección de lista a un marco.
// Suponga que "frame" es un objeto JFrame
// creado en otra parte.
```

```
frame.getContentPane ().add (listPane);
```

Ejemplo

Presentar una lista de archivos en spool utilizando como recurso de impresión el objeto VPrinterOutput. La figura 1 muestra el componente de la interfaz gráfica de usuario VPrinterOutput:

Figura 1: componente GUI VPrinterOutput



Clase SpooledFileViewer

La clase SpooledFileViewer crea una ventana que permite ver los archivos AFP (funciones avanzadas de impresión) y SCS (serie de caracteres de arquitectura de red de sistemas) que se han enviado al spool de impresión. Esencialmente, la clase añade una función de "presentación preliminar" a los archivos en spool (función habitual en la mayoría de los programas de tratamiento de texto) como se muestra en la figura 1.

El visor de archivos en spool es de especial utilidad cuando examinar la exactitud del diseño de los archivos sea más importante que imprimir los archivos, cuando ver los datos sea más rentable que imprimirlos, o cuando no haya ninguna impresora disponible.

Nota: es preciso instalar SS1 Opción 8 (Compatibilidad de fonts AFP) en el servidor de sistema principal.

Utilización de la clase SpooledFileViewer

Hay disponibles tres métodos constructores para crear una instancia de la clase SpooledFileViewer. Puede emplearse el constructor SpooledFileViewer() para crear un visor que no tenga asociado ningún archivo en spool. De utilizarse este constructor, más adelante será preciso establecer un archivo en spool mediante setSpooledFile(SpooledFile). El constructor SpooledFileViewer(SpooledFile) permite crear un visor para el archivo en spool dado, siendo la página número uno la vista inicial. Por último, puede emplearse el constructor SpooledFileViewer(spooledFile, int) para crear un visor para el archivo en spool dado con la página especificada como la vista inicial. Independientemente del constructor utilizado, tras crear el visor es preciso realizar una llamada a load() para recuperar realmente los datos del archivo en spool.

Entonces, el programa puede ir recorriendo las páginas individuales del archivo en spool mediante estos métodos:

- load FlashPage()
- load Page()
- pageBack()
- pageForward()

Sin embargo, si necesita examinar más detenidamente algunas secciones del documento, puede agrandar o reducir la imagen de una página del documento, alterando los factores de proporción de cada página con estos métodos:

- fitHeight()
- fitPage()
- fitWidth()
- actualSize()

El programa concluiría con una llamada al método close() que cierra la corriente de entrada y libera los recursos asociados a la corriente.

Utilización de SpooledFileViewer

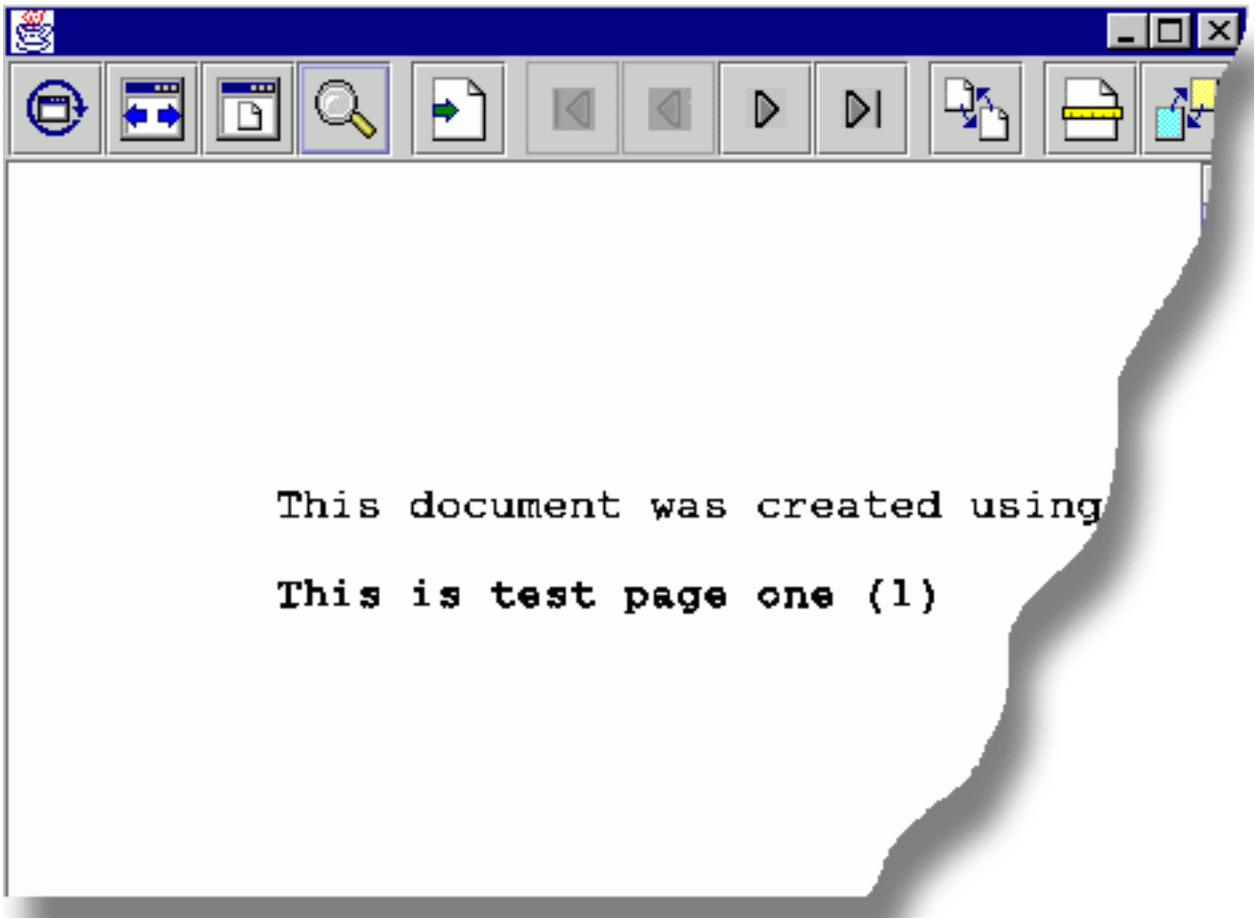
Una instancia de la clase SpooledFileViewer es en realidad una representación gráfica de un visor capaz de visualizar y navegar por un archivo en spool AFP o SCS. Por ejemplo, el código que hay a continuación crea el visor de archivos en spool de la figura 1 para visualizar un archivo en spool creado previamente en el servidor.

Nota: puede seleccionar un botón de la imagen en la figura 1 para obtener una descripción de su función o (si el navegador no está habilitado para JavaScript) consultar la descripción de la barra de herramientas.

```
// Supongamos que splf es el archivo en spool.
```

```
// Cree el visor de archivos en spool.
SpooledFileViewer splfv = new SpooledFileViewer(splf, 1);
splfv.load();
// Añada el visor de archivos en spool a un marco.
JFrame frame = new JFrame("Mi ventana");
frame.getContentPane().add(splfv);
frame.pack();
frame.show();
```

Figura 1: SpooledFileViewer



SpooledFileViewer: descripción de la barra de herramientas


El botón "tamaño real" hace que la imagen de la página del archivo en spool recupere su tamaño original utilizando el método actualSize().


El botón "ajustar anchura" hace que la imagen de la página del archivo en spool se ajuste a los bordes izquierdo y derecho del marco del visor utilizando el método fitWidth().


El botón "ajustar página" hace que la imagen de la página del archivo en spool se ajuste vertical y horizontalmente al marco del visor del archivo en spool utilizando el método fitPage().

 El botón "zoom" le permite aumentar o reducir el tamaño de la imagen en la página del archivo en spool; para ello, seleccione uno de los porcentajes predefinidos o escriba el porcentaje que desee en un campo de texto que aparece en un cuadro de diálogo tras seleccionar el botón "zoom".

 El botón "ir a página", si lo selecciona, le permite ir a una página específica del archivo en spool.

 El botón "primera página", si lo selecciona, le lleva a la primera página del archivo en spool y, desactivado, le indica que está en la primera página.

 El botón "página anterior", si lo selecciona, le lleva a la página que hay justo antes de la que visualiza en este momento.

 El botón "página siguiente", si lo selecciona, le permite avanzar a la página que figura justo después de la que visualiza en este momento.

 El botón "última página", si lo selecciona, le permite avanzar hasta la última página del archivo en spool y, desactivado, le indica que está en la última página.

 El botón "cargar página de memoria flash", cuando se selecciona, carga la página visualizada anteriormente utilizando el método loadFlashPage().

 El botón "establecer tamaño de papel", cuando se selecciona, permite establecer el tamaño del papel.

 El botón "establecer fidelidad de visualización", cuando se selecciona, permite establecer la fidelidad de visualización.

Clases de llamada a mandato de vaccess

Los componentes de llamada a programa del paquete vaccess permiten a un programa Java presentar un botón o un elemento de menú que llama a un programa servidor. Pueden especificarse parámetros de entrada, salida y entrada/salida mediante los objetos ProgramParameter. Cuando el programa se ejecuta, los parámetros de salida y entrada/salida contienen los datos devueltos por el programa servidor.

Un objeto ProgramCallButton representa un botón que, cuando se pulsa, llama a un programa servidor. La clase ProgramCallButton amplía la clase JButton JFC (clases Java fundamentales) para que todos los botones tengan un aspecto y un comportamiento coherentes.

De forma parecida, un objeto ProgramCallMenuItem representa un elemento de menú que, cuando se selecciona, llama a un programa servidor. La clase ProgramCallMenuItem amplía la clase JMenuItem JFC para que todos los elementos de menú tengan también un aspecto y un comportamiento coherentes.

Para utilizar un componente de llamada a programa de vaccess, establezca las propiedades system y program. Estas propiedades se establecen mediante un constructor o con los métodos setSystem() y setProgram().

El siguiente ejemplo crea un objeto ProgramCallMenuItem. En tiempo de ejecución, el elemento de menú, cuando se selecciona, llama a un programa:

```
        // Cree el objeto ProgramCallMenuItem.
        // Suponga que "system" es
        // un objeto AS400 creado e
        // inicializado en otra parte. El texto del
        // elemento de menú indica "Seleccione esto"
        // y no hay ningún icono.
    ProgramCallMenuItem menuItem = new ProgramCallMenuItem ("Seleccione
esto", null, system);

        // Cree un objeto nombre de vía de acceso que
        // represente el programa MYPROG de la
        // biblioteca MYLIB.
    QSYSObjectPathName programName = new QSYSObjectPathName("MYLIB",
"MYPROG", "PGM");

        // Establezca el nombre del programa.
    menuItem.setProgram (programName.getPath());

        // Añada el elemento a un menú.
        // Supongamos que el menú se ha creado
        // en otra parte.
    menu.add (menuItem);
```

Un programa servidor, cuando se ejecuta, puede devolver cero o más mensajes del servidor. Para detectar cuándo se ejecuta el programa servidor, añada un escuchador ActionCompletedListener al botón o al elemento de menú, utilizando para ello el método addActionCompletedListener(). El programa, cuando se ejecuta, activa un evento ActionCompletedEvent para todos estos escuchadores. Un escuchador puede utilizar el método getMessageList() para recuperar los mensajes del servidor que el programa haya generado.

Este ejemplo añade un escuchador ActionCompletedListener que procesa todos los mensajes del servidor generados por el programa:

```
        // Añada un ActionCompletedListener
        // que se implementa utilizando una clase
        // interna anónima. Es un modo práctico
        // de especificar escuchadores de
```

```
        // eventos simples.
     menuItem.addActionCompletedListener (new ActionCompletedListener ()
     {
          public void actionCompleted (ActionCompletedEvent event)
          {
        // Convierta el origen del evento en un
        // ProgramCallMenuItem.
             ProgramCallMenuItem sourceMenuItem = (ProgramCallMenuItem)
event.getSource ();

        // Obtenga la lista de mensajes del servidor
        // que el programa ha generado.
             AS400Message[] messageList = sourceMenuItem.getMessageList
();

        // ...Procese la lista de mensajes.
          }
     });
```

Parámetros

Los objetos ProgramParameter permiten pasar datos de parámetro entre el programa Java y el programa servidor. Los datos de entrada se establecen con el método setInputData(). Tras la ejecución del programa, los datos de salida se recuperan con el método getOutputData().

Cada parámetro es una matriz de bytes. Es el programa Java el que se ocupa de convertir la matriz de bytes entre el formato Java y el formato del servidor. Las clases de conversión de datos proporcionan métodos para convertir los datos.

A un componente de la interfaz gráfica de usuario de llamada a programa se le pueden añadir parámetros ya sea de uno en uno, utilizando el método addParameter(), o todos de una vez, mediante el método setParameterList().

Encontrará más información acerca de cómo se utilizan los objetos ProgramParameter en la clase de acceso a ProgramCall.

El siguiente ejemplo añade dos parámetros:

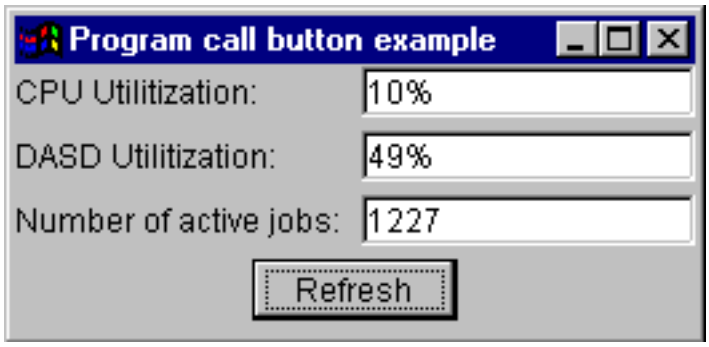
```
        // El primer parámetro es un nombre de tipo serie
        // de hasta 100 caracteres.
        // Es un parámetro de entrada.
        // Suponga que "name" es un objeto String
        // creado e inicializado en otra parte.
     AS400Text parm1Converter = new AS400Text (100, system.getCcsid (),
system);
     ProgramParameter parm1 = new ProgramParameter (parm1Converter.toBytes
(name));
     menuItem.addParameter (parm1);

        // El segundo es un parámetro de salida
        // de tipo entero.
     AS400Bin4 parm2Converter = new AS400Bin4 ();
     ProgramParameter parm2 = new ProgramParameter
(parm2Converter.getByteLength ());
     menuItem.addParameter (parm2);

        // ...Tras llamar al programa,
        // obtenga el valor devuelto como
        // segundo parámetro.
```

```
int result = parm2Converter.toInt (parm2.getOutputData ());
```

Ejemplos

Ejemplo de cómo se utiliza un objeto ProgramCallButton en una aplicación.La figura 1 muestra el aspecto de ProgramCallButton:

Figura 1: cómo se utiliza ProgramCallButton en una aplicación



Clases de acceso a nivel de registro de vaccess

Las clases de acceso a nivel de registro del paquete vaccess permiten a un programa Java presentar diversas vistas de los archivos de servidor.

Los componentes disponibles son los siguientes:

- RecordListFormPane - Presenta en un formulario una lista de registros de un archivo de servidor.
- RecordListTablePane - Presenta en una tabla una lista de registros de un archivo de servidor.
- RecordListTableModel - Gestiona la lista de registros de un archivo de servidor para una tabla.

Acceso por clave

Los componentes de la interfaz gráfica de usuario de acceso a nivel de registro se pueden utilizar con el acceso por clave a un archivo de servidor. Acceso por clave quiere decir que el programa Java tan solo ha de especificar una clave para acceder a los registros de un archivo.

El acceso por clave funciona de modo idéntico para cada componente de la interfaz gráfica de usuario del acceso a nivel de registro. Utilice setKeyed() para especificar que se realiza el acceso por clave, en vez del acceso secuencial. La clave se especifica mediante el constructor o con el método setKey(). En Especificación de la clave encontrará más información acerca de cómo se especifica la clave.

Por omisión, solo se visualizan los registros cuyas claves sean iguales a la clave especificada. Si desea cambiar este comportamiento por omisión, especifique la propiedad searchType mediante el constructor o con el método setSearchType(). Las opciones posibles son las siguientes:

- KEY_EQ - Visualizar los registros cuyas claves sean iguales a la especificada.
- KEY_GE - Visualizar los registros cuyas claves sean mayores o iguales que la especificada.
- KEY_GT - Visualizar los registros cuyas claves sean mayores que la especificada.
- KEY_LE - Visualizar los registros cuyas claves sean menores o iguales que la especificada.
- KEY_LT - Visualizar los registros cuyas claves sean menores que la especificada.

El siguiente ejemplo crea un objeto RecordListTablePane para visualizar todos los registros cuyas claves sean menores o iguales que una clave.

```
        // Cree una clave que contenga un
        // solo elemento, el entero (Integer) 5.
    Object[] key = new Object[1];
    key[0] = new Integer (5);

        // Cree un objeto RecordListTablePane.
        // Suponga que "system" es
        // un objeto AS400 creado e
        // inicializado en otra parte. Especifique
        // la clave y el tipo de búsqueda.
    RecordListTablePane tablePane = new RecordListTablePane (system,
         "/QSYS.LIB/QGPL.LIB/PARTS.FILE", key, RecordListTablePane.KEY_LE);

        // Cargue el contenido del archivo.
    tablePane.load ();

   // Añada la sección tabla a un marco.
   // Suponga que "frame" es un objeto JFrame
   // creado en otra parte.
    frame.getContentPane ().add (tablePane);
```

Clase RecordListFormPane

Una sección RecordListFormPane presenta el contenido de un archivo de servidor en un formulario. El formulario visualiza los registros de uno en uno y proporciona botones que permiten al usuario desplazarse hacia delante, hacia atrás, al primer registro o al último registro, o bien renovar la vista del contenido del archivo.

Para utilizar un objeto RecordListFormPane, establezca las propiedades system y fileName. Estas propiedades se establecen mediante el constructor o con los métodos setSystem() y setFileName(). Utilice load() para recuperar el contenido del archivo y presentar el primer registro. Cuando ya no necesite el contenido del archivo, llame a close() para asegurarse de que se cierre el archivo.

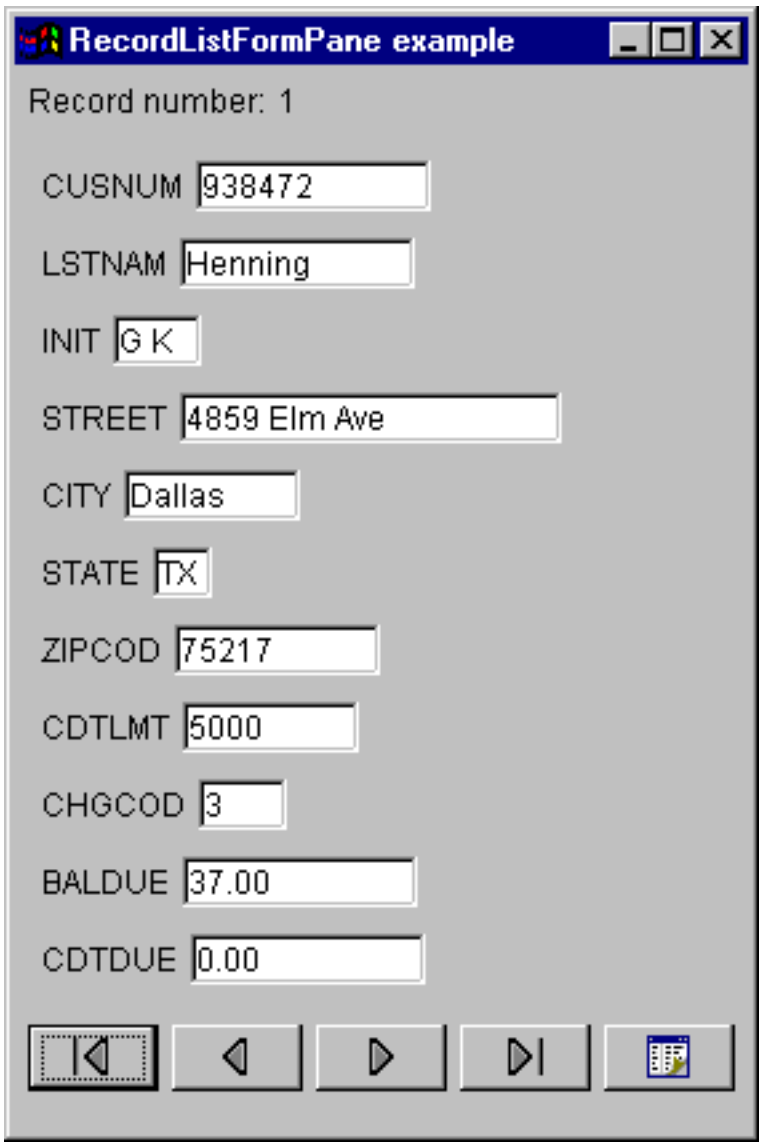
El siguiente ejemplo crea un objeto RecordListFormPane y lo añade a un marco:

```
         // Cree un objeto RecordListFormPane.
         // Suponga que "system" es
         // un objeto AS400 creado e
         // inicializado en otra parte.
     RecordListFormPane formPane = new RecordListFormPane (system,
"/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

         // Cargue el contenido del archivo.
     formPane.load ();

    // Añada la sección formulario a un marco.
    // Suponga que "frame" es un objeto JFrame
    // creado en otra parte.
     frame.getContentPane ().add (formPane);
```

Ejemplo

Presentar una sección RecordListFormPane que visualiza el contenido de un archivo.La figura 1 muestra el componente de la interfaz gráfica de usuario RecordListFormPane:

Figura 1: componente GUI RecordListFormPane


RecordListFormPane example
Record number: 1
CUSNUM 938472
LSTNAM Henning
INIT G K
STREET 4859 Elm Ave
CITY Dallas
STATE TX
ZIPCOD 75217
CDTLMT 5000
CHGCOD 3
BALDUE 37.00
CDTDUE 0.00

Clase RecordListTablePane

Una sección RecordListTablePane presenta en una tabla el contenido de un archivo de servidor. Cada fila de la tabla visualiza un registro del archivo y cada columna visualiza un campo.

Para utilizar un objeto RecordListTablePane, establezca las propiedades system y fileName. Estas propiedades se establecen mediante el constructor o los métodos setSystem() y setFileName(). Utilice load() para recuperar el contenido del archivo y presentar los registros en la tabla. Cuando ya no necesite el contenido del archivo, llame a close() para asegurarse de que se cierre el archivo.

El siguiente ejemplo crea un objeto RecordListTablePane y lo añade a un marco:

```
         // Cree un objeto RecordListTablePane.
         // Supongamos que "system" es
         // un objeto AS400 creado e
         // inicializado en otra parte.
     RecordListTablePane tablePane = new RecordListTablePane (system,
"/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

         // Cargue el contenido del archivo.
     tablePane.load ();

         // Añada la sección tabla a un marco.
    // Supongamos que "frame" es un objeto JFrame
    // creado en otra parte.
     frame.getContentPane ().add (tablePane);
```

Clases RecordListTablePane y RecordListTableModel

La sección RecordListTablePane se implementa mediante el paradigma controlador de modelos-vistas, en el que los datos y la interfaz de usuario están separados en distintas clases. La implementación integra RecordListTableModel en la clase JTable de JFC (clases Java fundamentales). La clase RecordListTableModel recupera y gestiona el contenido del archivo, y la clase JTable visualiza gráficamente el contenido del archivo y gestiona la interacción de usuario.

RecordListTablePane proporciona suficientes funciones para la mayoría de las necesidades. Sin embargo, un llamador, si necesita un mayor control del componente JFC, puede utilizar directamente la clase RecordListTableModel y proporcionar una integración personalizada en otro componente de la interfaz gráfica de usuario.

Para utilizar un objeto RecordListTableModel, establezca las propiedades system y fileName. Estas propiedades se establecen mediante el constructor o con los métodos setSystem() y setFileName(). Utilice load() para recuperar el contenido del archivo. Cuando ya no necesite el contenido del archivo, llame a close() para asegurarse de que se cierre el archivo.

El siguiente ejemplo crea un objeto RecordListTableModel y lo presenta en una tabla JTable:

```
         // Cree un objeto RecordListTableModel.
         // Supongamos que "system" es
         // un objeto AS400 creado e
         // inicializado en otra parte.
     RecordListTableModel tableModel = new RecordListTableModel (system,
"/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

         // Cargue el contenido del archivo.
     tableModel.load ();

         // Cree una tabla JTable para el modelo.
     JTable table = new JTable (tableModel);

         // Añada la tabla a un marco. Supongamos
         // que "frame" es un objeto JFrame
    // creado en otra parte.
     frame.getContentPane ().add (table);
```

ResourceListPane y ResourceListDetailsPane

Utilice las clases ResourceListPane y ResourceListDetailsPane para presentar una lista de recursos en una interfaz gráfica de usuario (GUI).

- ResourceListPane visualiza el contenido de la lista de recursos en un objeto javax.swing.JList gráfico. Cada uno de los elementos de la lista representa un objeto de recurso de la lista de recursos.
- ResourceListDetailsPane visualiza el contenido de la lista de recursos en un objeto javax.swing.JTable gráfico. Cada una de las filas de la tabla representa un objeto de recurso de la lista de recursos.

Las columnas de tabla de un objeto ResourceListDetailsPane se especifican como una matriz de ID de atributo de columna. La tabla contiene una columna para cada elemento de la matriz y una fila para cada objeto de recurso.

Por omisión los menús emergentes están habilitados tanto para ResourceListPane como para ResourceListDetailsPane.

La mayor parte de los errores se reportan como com.ibm.as400.vaccess.ErrorEvents en lugar de como excepciones lanzadas. Escuche los objetos ErrorEvent para llevar a cabo las tareas de diagnóstico y recuperación de condiciones de error.

Ejemplo: visualizar una lista de recursos en una GUI

Este ejemplo crea un objeto ResourceList de todos los usuarios de un sistema y lo visualiza en una GUI (sección de detalles):

```
   // Cree la lista de recursos.
   AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
   RUserList userList = new RUserList(system);

   // Cree el objeto ResourceListDetailsPane. En este ejemplo,
   // hay dos columnas en la tabla. La primera columna
   // contiene los iconos y los nombres de cada usuario. La
   // segunda columna contiene la descripción de texto de cada
   // usuario.
   Object[] columnAttributeIDs = new Object[] { null, RUser.TEXT_DESCRIPTION
};
   ResourceListDetailsPane detailsPane = new ResourceListDetailsPane();
   detailsPane.setResourceList(userList);
   detailsPane.setColumnAttributeIDs(columnAttributeIDs);

   // Añada el objeto ResourceListDetailsPane a un objeto JFrame y
muéstrelo.
JFrame frame = new JFrame("Mi ventana");
     frame.getContentPane ().add (detailsPane);
   frame.pack();
   frame.show();

   // El objeto ResourceListDetailsPane aparecerá vacío hasta
   // que lo carguemos. Esto nos permite controlar cuándo se
   // recupera la lista de usuarios del iSeries.
            detailsPane.load ();
```

Clases del estado del sistema

Los componentes de estado del sistema del paquete vaccess permiten crear interfaces GUI utilizando los objetos AS400Pane existentes. También puede optar por crear sus propias GUI utilizando las clases JFC (clases Java fundamentales). El objeto VSystemStatus representa un estado del sistema en el servidor. El objeto VSystemPool representa una agrupación del sistema en el servidor. El objeto VSystemStatusPane representa una sección visual que muestra información del estado del sistema.

La clase VSystemStatus permite obtener información acerca del estado de una sesión de servidor dentro de un entorno GUI:

- El método getSystem() devuelve el servidor en el que se encuentra la información del estado del sistema.
- El método getText() devuelve el texto descriptivo.
- El método setSystem() establece el servidor en el que se ubica la información de estado del sistema.

Además de los métodos que acaban de mencionarse, también es posible acceder a información de agrupación del sistema y realizar cambios en dicha información en una GUI.

El objeto VSystemStatus se utiliza junto con la sección VSystemStatusPane. VSystemPane es la sección de pantalla visual en la que se muestra información tanto del estado del sistema como de la agrupación del sistema.

Clase VSystemStatusPane

La clase VSystemStatusPane permite a un programa Java visualizar información de estado del sistema y de agrupación del sistema.

La sección VSystemStatusPane incluye estos métodos:

- getVSystemStatus(): devuelve la información de VSystemStatus en una sección VSystemStatusPane.
- setAllowModifyAllPools(): establece el valor para determinar si puede modificarse la información de agrupación del sistema.

El ejemplo siguiente muestra cómo se utiliza la clase VSystemStatusPane:

```
// Cree un objeto as400.
AS400 mySystem = new AS400("mySystem.myCompany.com");

// Cree una sección VSystemStatusPane
VSystemStatusPane myPane = new VSystemStatusPane(mySystem);

// Establezca el valor que permite modificar las agrupaciones.
myPane.setAllowModifyAllPools(true);

//Cargue la información.
myPane.load();
```

GUI de valores del sistema

Los componentes de valores del sistema del paquete vaccess permiten a un programa Java crear interfaces GUI utilizando los objetos AS400Pane existentes o creando sus propias secciones utilizando las clases JFC (clases Java fundamentales). El objeto VSystemValueList representa una lista de valores del sistema en el servidor.

Para utilizar el componente GUI de valores del sistema, establezca el nombre del sistema con un constructor o mediante el método setSystem().

Ejemplo

El siguiente ejemplo crea una GUI de valores del sistema utilizando una sección de panel explorador de AS/400 (AS400ExplorerPane):

```
//Cree un objeto AS400.
AS400 mySystem = newAS400("mySystem.myCompany.com");
VSystemValueList mySystemValueList = new VSystemValueList(mySystem);
as400Panel=new AS400ExplorerPane((VNode)mySystemValueList);
//Cree y cargue un objeto AS400ExplorerPane.
as400Panel.load();
```

Clases de usuarios y grupos de vaccess

Los componentes de usuarios y grupos del paquete vaccess permiten presentar listas de usuarios y grupos del servidor mediante la clase VUser.

Los componentes disponibles son los siguientes:

- Las secciones de panel AS/400 (AS400Pane) son componentes de la interfaz gráfica de usuario que presentan uno o varios recursos del servidor y permiten su manipulación.
- Un objeto VUserList es un recurso que representa una lista de usuarios y grupos del servidor para utilizarse en los objetos AS400Pane.
- Un objeto VUserAndGroup es un recurso que representa grupos de usuarios del servidor para utilizarse en los objetos AS400Pane. Permite a un programa Java listar todos los usuarios, listar todos los grupos o listar los usuarios que no forman parte de ningún grupo.

Los objetos AS400Pane y VUserList se pueden utilizar conjuntamente para presentar múltiples vistas de la lista. También se pueden emplear para permitir al usuario seleccionar usuarios y grupos.

Para utilizar una lista VUserList, primero es necesario establecer la propiedad system. Esta propiedad se establece mediante un constructor o con el método setSystem(). Posteriormente, el objeto VUserList se "conecta" al objeto AS400Pane como raíz, utilizando el constructor o el método setRoot() del objeto AS400Pane.

VUserList dispone de otras propiedades que son de utilidad para definir el conjunto de usuarios y grupos presentados en los objetos AS400Pane:

- Utilice el método setUserInfo() para especificar los tipos de usuarios que deben mostrarse.
- Utilice el método setGroupInfo() para especificar un nombre de grupo.

Puede utilizar el objeto VUserAndGroup para obtener información acerca de los usuarios y grupos del sistema. Para obtener información acerca de un determinado objeto, primero es preciso cargar la información a fin de que se pueda acceder a ella. Si desea visualizar el servidor en el que se encuentra la información, utilice el método getSystem.

Los objetos AS400Pane y VUserList o VUserAndGroup, cuando se crean, se inicializan en un estado por omisión. La lista de usuarios y grupos no se ha cargado. Para cargar el contenido, el programa Java debe llamar explícitamente al método load() en cualquiera de los objetos para iniciar la comunicación con el servidor a fin de recopilar el contenido de la lista.

En tiempo de ejecución, pulse el botón derecho del ratón en un usuario, una lista de usuarios o un grupo para visualizar el menú de atajo. Seleccione **Propiedades** en el menú de atajo para llevar a cabo acciones en el objeto seleccionado:

- Usuario - visualiza una lista de información de usuario con estos elementos: descripción, clase de usuario, estado, descripción de trabajo, información de salida, información de mensaje, información internacional, información de seguridad e información de grupo.
- Lista de usuarios - permite trabajar con las propiedades de información de usuario e información de grupo. También puede cambiar el contenido de la lista.
- Usuarios y grupos - visualiza propiedades tales como el nombre de usuario y la descripción.

Los usuarios únicamente pueden acceder a los usuarios y grupos sobre los que poseen autorización. Además, el programa Java puede impedir que el usuario realice determinadas acciones, utilizando para ello el método setAllowActions() en la sección.

El siguiente ejemplo crea una lista VUserList y la presenta en una sección AS400DetailsPane:

```
// Cree el objeto VUserList.
// Supongamos que "system" es un objeto
// AS400 creado e inicializado
```

```
    // en otra parte.
    VUserList root = new VUserList (system);

    // Cree y cargue un objeto
    // AS400DetailsPane.
     AS400DetailsPane detailsPane = new AS400DetailsPane (root);
            detailsPane.load ();

    // Añada la sección de detalles a un marco.
    // Supongamos que "frame" es un objeto JFrame
    // creado en otra parte.
     frame.getContentPane ().add (detailsPane);
```

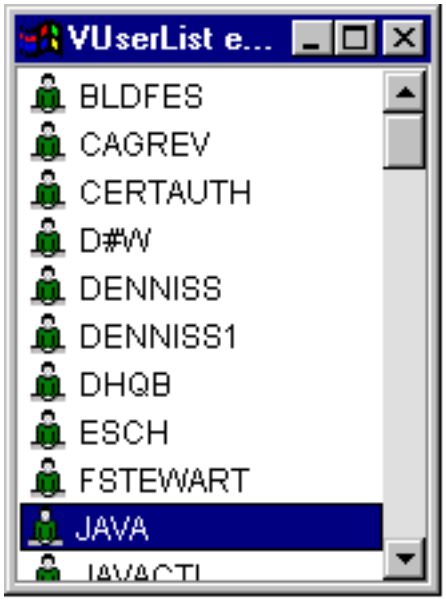
El ejemplo siguiente muestra cómo se utiliza el objeto VUserAndGroup:

```
    // Cree el objeto VUserAndGroup.
    // Supongamos que "system" es un objeto AS400 creado e inicializado en
otra parte.
    VUserAndGroup root = new VUserAndGroup(system);

    // Cree y cargue un objeto AS400ExplorerPane.
     AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
                    explorerPane.load ();

    // Añada la sección de panel explorador a un marco.
    // Supongamos que "frame" es un objeto JFrame creado en otra parte.
     frame.getContentPane ().add (explorerPane);
```

Otros ejemplos

Presentar una lista de usuarios existentes en el sistema utilizando una sección AS400ListPane con un objeto VUserList.

La imagen siguiente muestra el componente de la interfaz gráfica de usuario de VUserList:



La Caja de Herramientas Gráfica

La Caja de Herramientas Gráfica es un conjunto de herramientas de interfaz de usuario (UI) gracias a las cuales resulta muy fácil crear paneles de interfaz de usuario personalizados en Java. Los paneles se pueden incorporar a las aplicaciones Java, a los applets o a los conectores de iSeries Navigator. Los paneles pueden contener datos obtenidos del iSeries o de cualquier otra fuente como, por ejemplo, un archivo del sistema de archivos local o un programa de la red.

El **Constructor de GUI** es un editor visual de tipo WYSIWYG (lo que se ve es lo que se obtiene) que permite crear diálogos, hojas de propiedades y asistentes Java. Con el Constructor de GUI puede añadir, organizar o editar controles de interfaz de usuario en un panel y después obtener una vista previa del panel para verificar si su diseño es el que esperaba. Las definiciones de panel creadas se pueden usar en diálogos, se pueden insertar en hojas de propiedades y asistentes o se pueden organizar en secciones divididas, secciones baraja y secciones con pestañas. El Constructor de GUI le permite asimismo construir barras de menús, barras de herramientas y definiciones de menú según contexto. También puede incorporar JavaHelp en sus paneles, incluyendo ayuda según contexto.

El **convertidor de scripts de recursos** convierte los scripts de recursos de Windows en una representación XML que los programas Java pueden utilizar. Gracias al convertidor de scripts de recursos, podrá procesar los scripts de recursos (archivos RC) de Windows desde los diálogos y los menús existentes en Windows. Luego, esos archivos convertidos se pueden editar con el Constructor de GUI. Es posible confeccionar hojas de propiedades y asistentes a partir de los archivos RC utilizando el convertidor de scripts de recursos junto con el Constructor de GUI.

Estas dos herramientas tienen subyacente una tecnología nueva llamada **PDML** (**Panel Definition Markup Language**). El lenguaje PDML se basa en XML (Extensible Markup Language) y define un lenguaje independiente de la plataforma para describir el diseño de los elementos de interfaz de usuario. Una vez definidos los paneles en PDML, puede utilizar la API de tiempo de ejecución proporcionada por la Caja de Herramientas Gráfica para visualizarlos. Para visualizar los paneles, la API interpreta el lenguaje PDML y representa la interfaz de usuario utilizando las clases JFC (clases Java fundamentales).

Ventajas de la Caja de Herramientas Gráfica

Escribir menos código y ahorrar tiempo

Con la Caja de Herramientas Gráfica, se pueden crear interfaces de usuario basadas en Java de forma rápida y sencilla. El Constructor de GUI permite controlar con precisión el diseño de los elementos de interfaz de usuario de los paneles. Puesto que el diseño está descrito en PDML, no es necesario desarrollar código Java para definir la interfaz de usuario, ni tampoco es preciso volver a compilar el código para efectuar cambios. Todo ello se traduce en una notable reducción del tiempo empleado para crear y mantener las aplicaciones Java. El convertidor de scripts de recursos permite migrar grandes cantidades de paneles de Windows a Java con facilidad y rapidez.

Ayuda personalizada

La definición de interfaces de usuario en PDML supone algunas ventajas adicionales. Como toda la información de un panel se consolida en un lenguaje de códigos formal, se pueden mejorar las herramientas para que lleven a cabo servicios adicionales en nombre del desarrollador. Por ejemplo, tanto el Constructor de GUI como el convertidor de scripts de recursos pueden generar esqueletos HTML para la ayuda en línea del panel. Usted decide qué temas de ayuda se necesitan, y los temas se construyen automáticamente a partir de sus exigencias. En el esqueleto de la ayuda se incorporan directamente códigos de ancla para los temas de ayuda y así el escritor de la ayuda queda libre para centrarse en desarrollar el contenido adecuado. El entorno de ejecución de la Caja de Herramientas Gráfica visualiza automáticamente el tema de ayuda correcto en respuesta a la petición de un usuario.

Integración automática de los paneles en el código

Además, el lenguaje PDML proporciona códigos que asocian cada uno de los controles de un panel a un atributo de un bean Java. Una vez que haya identificado las clases bean que suministrarán los datos al panel y que haya asociado un atributo a cada uno de los controles adecuados, puede solicitar que las herramientas generen esqueletos de código fuente Java para los objetos bean. En el momento de la ejecución, la Caja de Herramientas Gráfica transfiere automáticamente los datos entre los beans y los controles existentes en el panel que usted haya identificado.

Independiente de plataforma

El entorno de ejecución de la Caja de Herramientas Gráfica proporciona soporte para el manejo de eventos, la validación de datos de usuario y los tipos comunes de interacción entre los elementos de un panel. Se establece automáticamente el aspecto correcto de la plataforma para la interfaz de usuario en función del sistema operativo subyacente; el Constructor de GUI permite conmutar entre distintas plataformas para ver el aspecto que ofrecerán los paneles a fin de que usted pueda evaluarlo.

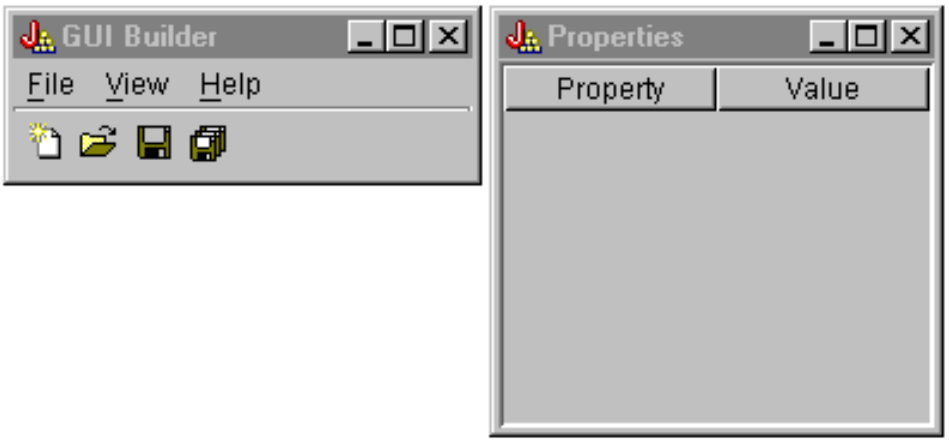
Dentro de la Caja de Herramientas Gráfica

La Caja de Herramientas Gráfica le proporciona dos herramientas y, en consecuencia, dos formas de automatizar la creación de sus propias interfaces de usuario. Puede utilizar el Constructor de GUI para crear paneles totalmente nuevos de forma rápida y sencilla o bien utilizar el convertidor de scripts de recursos para convertir a Java los paneles existentes basados en Windows. Los archivos convertidos se pueden editar posteriormente con el Constructor de GUI. Ambas herramientas soportan la internacionalización.

Constructor de GUI

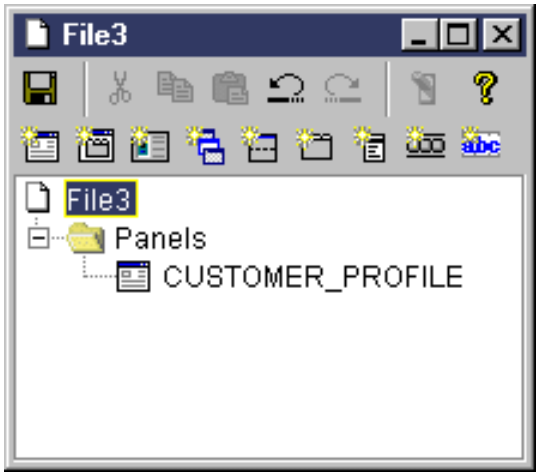
Al invocar el Constructor de GUI por primera vez, se visualizan dos ventanas, tal como se muestra en la figura 1:

Figura 1: ventanas del Constructor de GUI



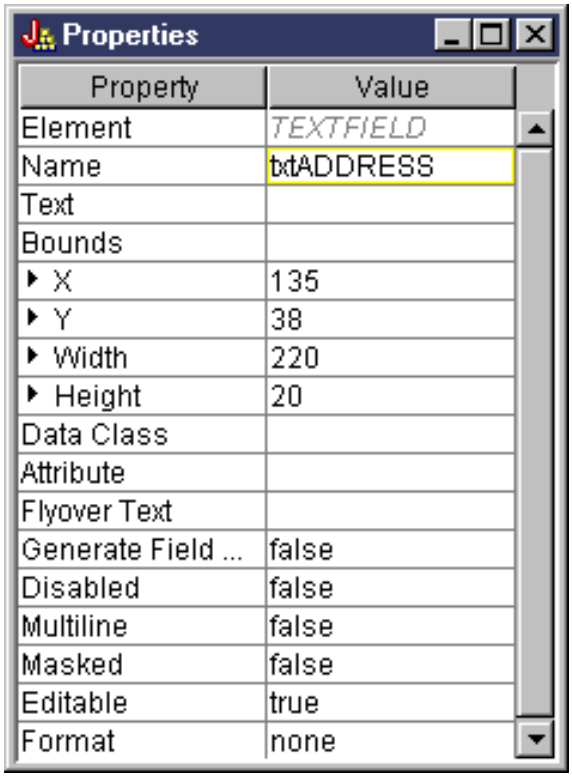
Utilice la ventana Constructor de archivos para crear y editar los archivos PDML.

Figura 2: ventana Constructor de archivos



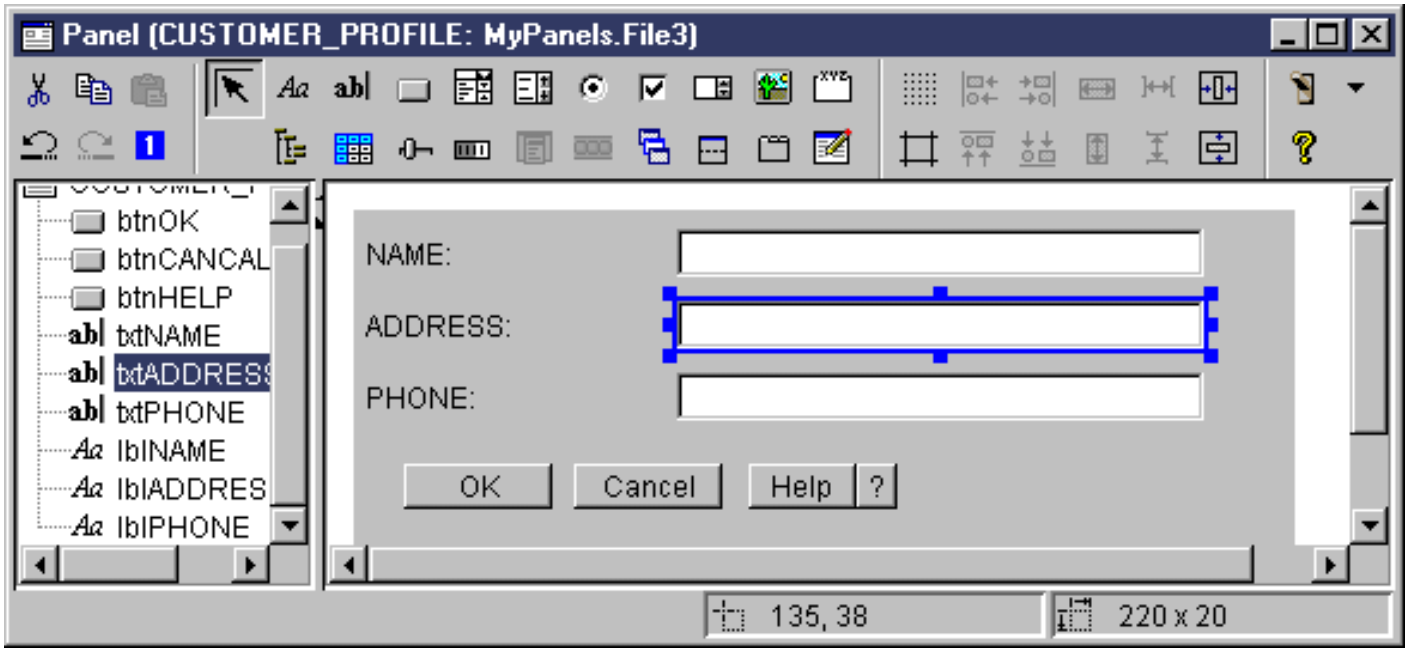
Utilice la ventana Propiedades para ver o cambiar las propiedades del control seleccionado en este momento.

Figura 3: ventana Propiedades



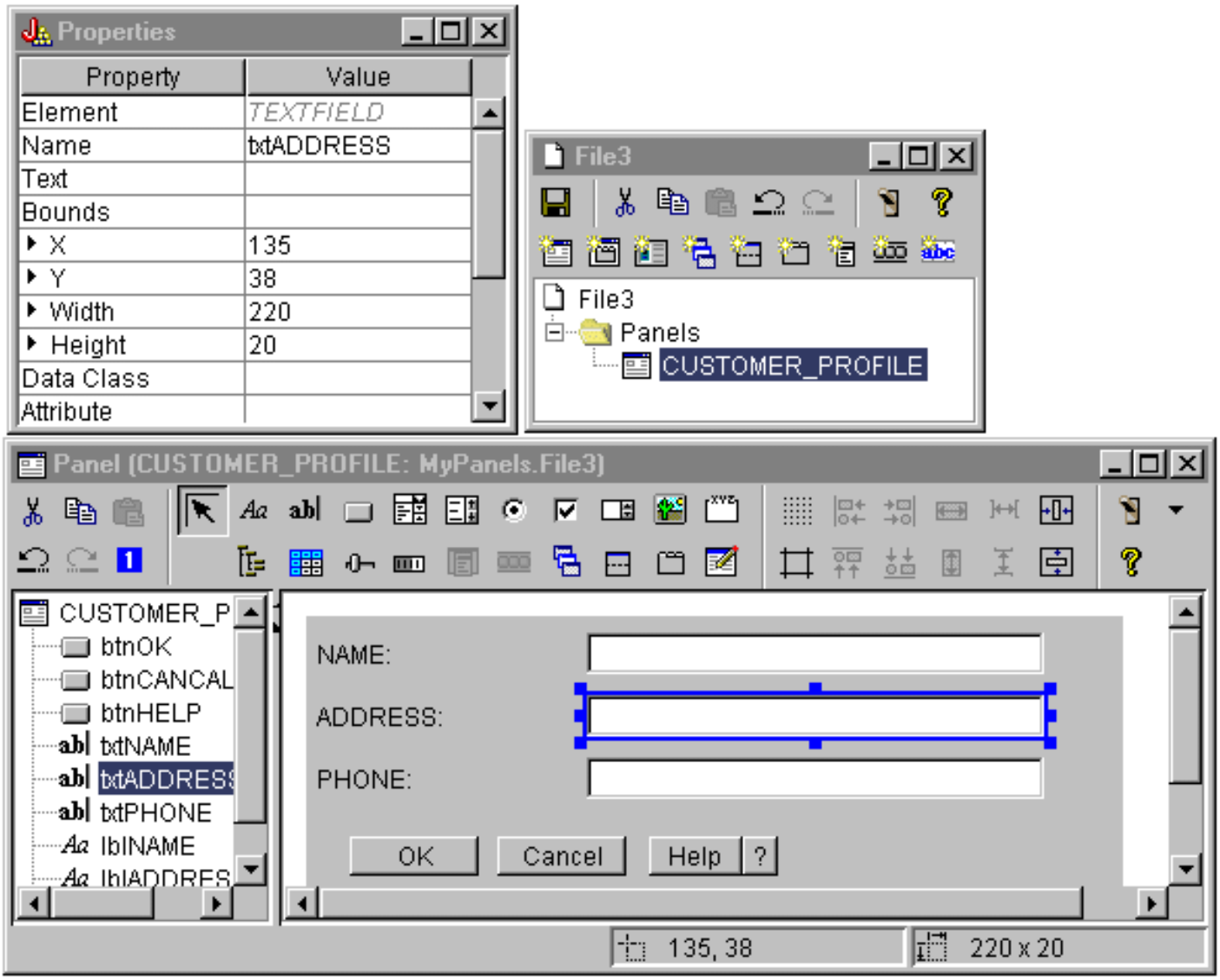
Utilice la ventana Constructor de paneles para crear y editar sus componentes de interfaz gráfica de usuario.Seleccione el componente que desee de la barra de herramientas y pulse en el panel para colocarlo en el lugar apetecido. La barra de herramientas también proporciona una serie de servicios que permiten alinear grupos de controles, obtener una vista previa del panel y solicitar la ayuda en línea correspondiente a una función del Constructor de GUI. En Barra de herramientas del constructor de paneles del Constructor de GUI encontrará una descripción de para qué sirve cada uno de los iconos.

Figura 4: ventana Constructor de paneles



El panel que se esté editando se visualiza en la ventana Constructor de paneles. La figura 5 muestra cómo funcionan las ventanas conjuntamente:

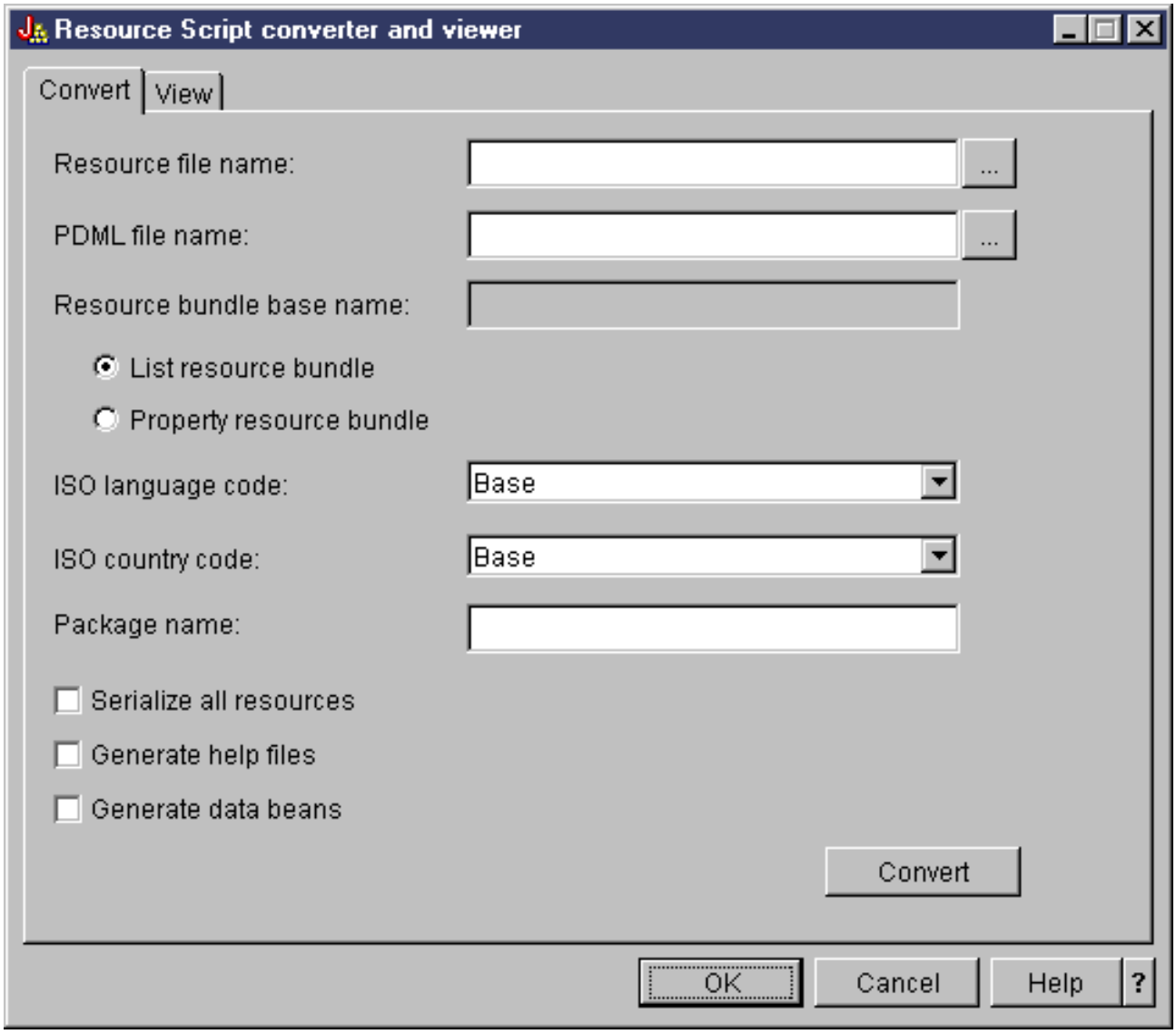
Figura 5: ejemplo de cómo funcionan conjuntamente las ventanas del Constructor de GUI



Convertidor de scripts de recursos

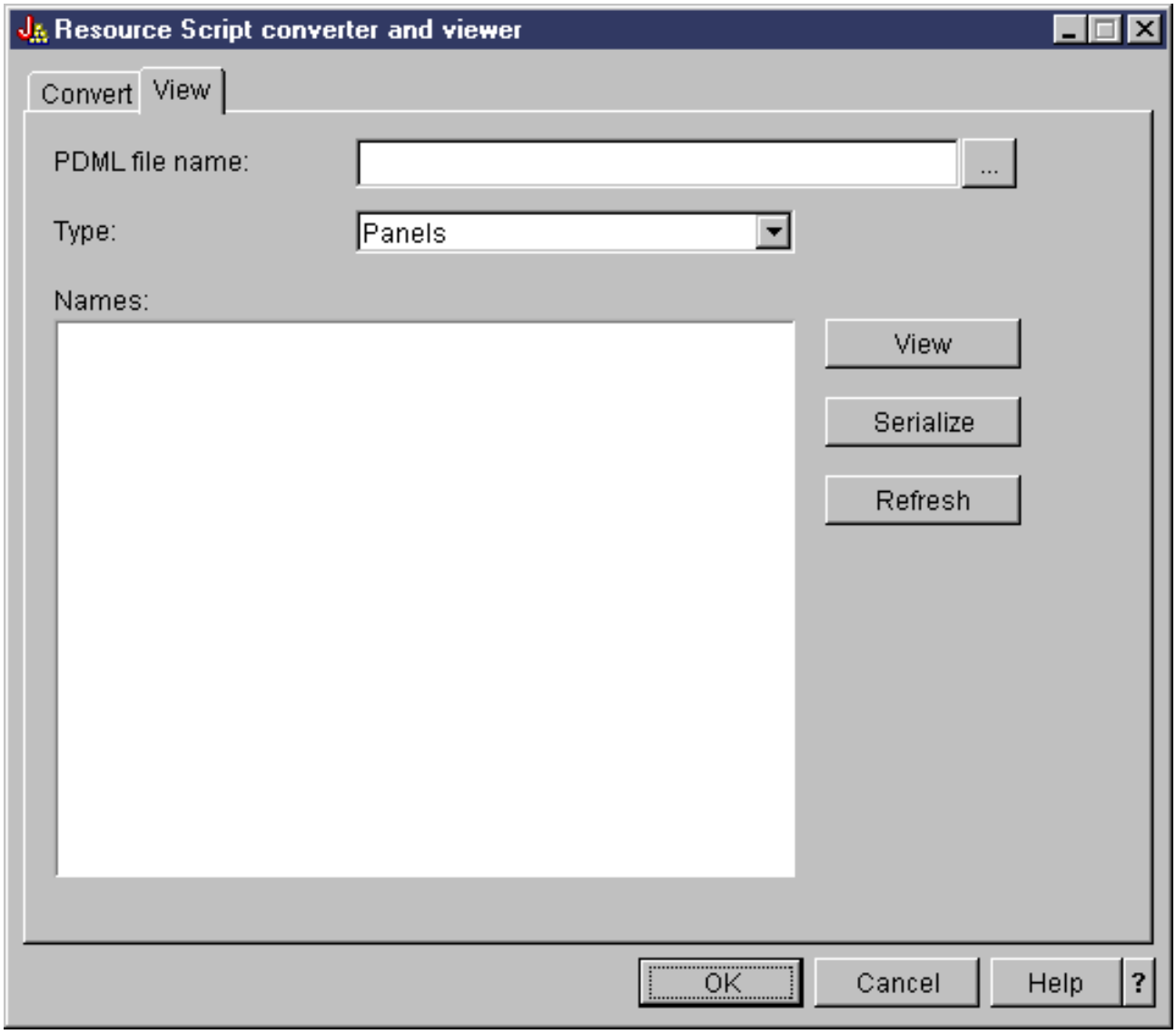
El convertidor de scripts de recursos es un diálogo formado por dos secciones con pestañas. En la sección **Convertir**, se especifica el nombre del archivo RC de Microsoft o VisualAge para Windows que se va a convertir a PDML. Puede especificar el nombre del archivo PDML destino y el paquete de recursos Java asociado que contendrán las series convertidas de los paneles. Además, puede solicitar que se generen los esqueletos de ayuda en línea para los paneles, así como generar esqueletos de código fuente Java para los objetos que suministran datos a los paneles y serializar las definiciones de los paneles para mejorar el rendimiento en tiempo de ejecución. La ayuda en línea del convertidor proporciona una descripción detallada de cada uno de los campos de entrada de la sección Convertir.

Figura 6: sección Convertir del convertidor de scripts de recursos



Una vez realizada la conversión de forma satisfactoria, puede utilizar la sección **Ver** para examinar el contenido del archivo PDML que acaba de crear, y obtener una vista previa de los nuevos paneles Java. Puede utilizar el Constructor de GUI para efectuar pequeños ajustes en un panel en caso de que sea necesario. El convertidor siempre comprueba si ya existe un archivo PDML antes de efectuar una conversión e intenta conservar los cambios por si más adelante necesita volver a ejecutar la conversión.

Figura 7: sección Ver del convertidor de scripts de recursos



Iniciación a la Caja de Herramientas Gráfica

Consulte estos temas si desea aprender más cosas sobre la Caja de Herramientas Gráfica:

- Puesta a punto de la Caja de Herramientas Gráfica
- Crear una interfaz de usuario propia
- Visualizar los paneles en tiempo de ejecución
- Generar archivos de ayuda en línea
- Ejemplo de la Caja de Herramientas Gráfica
- Utilización de la Caja de Herramientas Gráfica en un navegador
- Barra de herramientas del constructor de paneles

Puesta a punto de la Caja de Herramientas Gráfica

La Caja de Herramientas Gráfica se entrega como una serie de archivos JAR. Para ponerla a punto, debe instalar los archivos JAR en la estación de trabajo y establecer la variable de entorno CLASSPATH.

Asegúrese de que la estación de trabajo cumpla los requisitos para ejecutar IBM Toolbox para Java.

Instalación de la Caja de Herramientas Gráfica en la estación de trabajo

Si desea desarrollar programas Java mediante la Caja de Herramientas Gráfica, primero debe instalar los archivos JAR de la Caja de Herramientas Gráfica en la estación de trabajo. Utilice uno de los métodos siguientes:

Transferir los archivos JAR

Nota: la siguiente lista indica algunos de los métodos que puede emplear para transferir los archivos JAR. El programa bajo licencia IBM Toolbox para Java debe estar instalado en el iSeries. Además, debe bajar el archivo JAR para JavaHelp, jhall.jar, del sitio Web de Sun JavaHelp.

- Puede utilizar FTP (no olvide transferir los archivos en modalidad binaria) y copiar en un directorio local de su estación de trabajo los archivos JAR del directorio **/QIBM/ProdData/HTTP/Public/jt400/lib**
- Utilice iSeries Access para Windows para correlacionar una unidad de red.
- Por último, para instalar los archivos JAR de la Caja de Herramientas Gráfica, puede utilizar la clase AS400ToolboxInstaller que viene con IBM Toolbox para Java; para ello, debe especificar el nombre de paquete OPNAV. Encontrará más información en Clases de instalación y actualización en cliente

Instalar los archivos JAR con iSeries Access para Windows

También puede instalar la Caja de Herramientas Gráfica al instalar iSeries Access para Windows. IBM Toolbox para Java ahora se distribuye como parte de iSeries Access para Windows. Si va a instalar iSeries Access para Windows por primera vez, elija la instalación personalizada y seleccione el componente **IBM Toolbox para Java** en el menú de instalación. Si ya tiene instalado iSeries Access para Windows, puede utilizar el programa de instalación selectiva para instalar este componente, si es que todavía no lo ha hecho.

Establecer la vía de acceso de clases

Para utilizar la Caja de Herramientas Gráfica, debe añadir estos archivos JAR a su variable de entorno CLASSPATH (o especificarlos en la opción `classpath` de la línea de mandatos).

Por ejemplo, si ha copiado los archivos en el directorio **C:\gtbox\lib** de la estación de trabajo, debe añadir los siguientes nombres de vía a la variable classpath (vía de acceso de clases):

```
C:\gtbox\lib\uitools.jar;
C:\gtbox\lib\jui400.jar;
C:\gtbox\lib\data400.jar;
C:\gtbox\lib\util400.jar;
C:\gtbox\lib\x4j400.jar;
C:\gtbox\lib\jhall.jar;
```

Si ha instalado la Caja de Herramientas Gráfica mediante iSeries Access para Windows, todos los archivos JAR (excepto jhall.jar) residirán en el directorio **\Archivos de programa\Ibm\Client Access\jt400\lib** de la unidad en la que haya instalado iSeries Access para Windows. iSeries Access para Windows instala jhall.jar en el directorio **\Archivos de programa\Ibm\Client Access\jre\lib**. Los nombres de vía de acceso de la vía de acceso de clases deben reflejar este valor.

Descripción de los archivos JAR

- **uitools.jar**: contiene las herramientas Constructor de GUI y Convertidor de scripts de recursos.
- **jui400.jar**: contiene la API de tiempo de ejecución de la Caja de Herramientas Gráfica. Los programas Java utilizan esta API para visualizar los paneles construidos por medio de las herramientas. Estas clases pueden redistribuirse con las aplicaciones.
- **data400.jar**: contiene la API de tiempo de ejecución para PCML (Program Call Markup Language). Los programas Java utilizan esta API para llamar a los programas de iSeries cuyos parámetros y valores de retorno se identifican mediante el lenguaje PCML. Estas clases pueden redistribuirse con las aplicaciones.
- **util400.jar**: contiene las clases de utilidades para formatear los datos de iSeries y manejar los mensajes de iSeries. Estas clases pueden redistribuirse con las aplicaciones.
- **x4j400.jar**: contiene el analizador XML utilizado por las clases de API para interpretar los documentos PDML y PCML.
- **jhall.jar**: contiene las clases de JavaHelp que visualizan la ayuda en línea y la ayuda según contexto para los paneles creados con el Constructor de GUI.

Nota: puede emplear las versiones internacionalizadas de las herramientas Constructor de GUI y Convertidor de scripts de recursos. Para ejecutar una versión que no sea en inglés de Estados Unidos, debe añadir a la instalación de la Caja de Herramientas Gráfica la versión correcta de **uitools.jar** que corresponda a su idioma y país o región. Estos archivos JAR están disponibles en el servidor iSeries en **/QIBM/ProdData/HTTP/Public/jt400/Mri29xx**, donde 29xx el código NLV de OS/400 de 4 dígitos correspondiente a su idioma y país o región. (Los nombres de los archivos JAR que están en los diversos directorios MRI29xx incluyen sufijos de 2 caracteres para el código de país o región e idioma Java.) Este archivo JAR adicional debe añadirse a la variable classpath (vía de acceso de clases) antes de **uitools.jar** en el orden de búsqueda.

Cómo se utiliza la Caja de Herramientas Gráfica

Una vez instalada la Caja de Herramientas Gráfica, siga estos enlaces para aprender a utilizar las herramientas:

- Utilización del Constructor de GUI
- Utilización del convertidor de scripts de recursos

Crear una interfaz de usuario propia

Ejecución del Constructor de GUI

Para iniciar el Constructor de GUI, invoque el intérprete de Java con el mandato siguiente:

```
java com.ibm.as400.ui.tools.GUI Builder [-plaf aspecto]
```

Si no ha establecido la variable de entorno CLASSPATH de modo que contenga los archivos JAR de la Caja de Herramientas Gráfica, tendrá que especificarlos en la línea de mandatos mediante la opción `classpath`. Consulte el apartado Puesta a punto de la Caja de Herramientas Gráfica.

Opciones `-plaf` *`aspecto`*

El aspecto de plataforma deseado. Esta opción le permite alterar temporalmente el aspecto por omisión establecido en función de la plataforma en la que están efectuando las tareas de desarrollo para así obtener una vista previa de los paneles y ver qué aspecto tendrían en las distintas plataformas de sistema operativo. Los valores de aspecto que se aceptan son los siguientes:

- Windows
- Metal
- Motif

Por ahora, el Constructor de GUI no da soporte a algunos atributos de aspecto adicionales soportados por Swing 1.1.

Tipos de recursos de interfaz de usuario

Cuando inicie el Constructor de GUI por primera vez, debe crear un nuevo archivo PDML. En la barra de menús de la ventana del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**. Una vez creado el nuevo archivo PDML, puede definir cualesquiera de los siguientes tipos de recursos de interfaz de usuario que deban incluirse en él:

Panel

El tipo de recurso básico. Describe un área rectangular dentro de la cual se organizan los elementos de la interfaz de usuario. Los elementos de interfaz de usuario pueden ser simples controles (por ejemplo, botones de selección o campos de texto, imágenes, animaciones o controles personalizados) o subpaneles más complejos (vea las siguientes definiciones de sección dividida, sección baraja o sección con pestañas). Un panel puede definir el diseño de una ventana o un diálogo autónomos o puede definir uno de los subpaneles incluido dentro de otro recurso de interfaz de usuario.

Menú

Ventana emergente que contiene una o varias acciones seleccionables, cada una de ellas representada mediante un texto (por ejemplo, "Cortar", "Copiar", "Pegar"). Se pueden definir para las acciones teclas nemotécnicas o aceleradoras, elementos de menú especiales que sean botones de selección o recuadros de selección, y también se pueden insertar separadores y submenús en cascada. Un recurso menú se puede utilizar como menú de contexto autónomo, como menú desplegable o bien él mismo puede especificar la barra de menús asociada a un recurso panel.

Barra de herramientas

Ventana formada por una serie de pulsadores, cada uno de los cuales representa una posible acción de usuario. Cada botón puede contener texto, un icono o las dos cosas. La barra de herramientas se puede definir como flotable, lo que permite al usuario arrastrarla fuera de un panel y soltarla en una ventana autónoma.

Hoja de propiedades

Ventana o diálogo autónomo formado por una serie de paneles con pestañas y por los botones Aceptar, Cancelar y Ayuda. Los recursos de panel definen el diseño de cada una de las ventanas con pestañas.

Asistente

Ventana o diálogo autónomo que consta de una serie de paneles que se visualizan al usuario en una secuencia

predefinida, con los botones Atrás, Siguiente, Cancelar, Terminar y Ayuda. La ventana de asistente también puede mostrar una lista de tareas a la izquierda de los paneles a modo de seguimiento de cómo progresa el usuario en el asistente.

Sección dividida

Subsección que consta de dos paneles separados por una barra divisoria. Los paneles pueden disponerse horizontal o verticalmente.

Sección con pestañas

Subsección que forma un control con pestañas. Este control se puede colocar dentro de otro panel, dentro de una sección dividida o dentro de una sección baraja.

Sección baraja

Subsección que consta de un conjunto de paneles. Estos paneles solo se pueden visualizar de uno en uno. Por ejemplo, en tiempo de ejecución, la sección baraja podría cambiar el panel visualizado en función la acción de un usuario determinado.

Tabla de series

Conjunto de recursos de tipo serie y los identificadores asociados a los recursos.

Archivos generados

Las series traducibles de un panel no se almacenan en el propio archivo PDML, sino en un paquete de recursos Java aparte. Las herramientas permiten especificar cómo está definido el paquete de recursos, ya sea como archivo PROPERTIES Java o como subclase ListResourceBundle. Una subclase ListResourceBundle es una versión compilada de los recursos traducibles, lo que supone una mejora en el rendimiento de la aplicación Java. No obstante, esto ralentizará el proceso de guardar del Constructor de GUI, ya que el paquete de recursos de lista (ListResourceBundle) se compilará en cada operación de guardar. Por consiguiente, es mejor que empiece con un archivo PROPERTIES (el valor por omisión) hasta que esté satisfecho con el diseño de la interfaz de usuario.

Puede utilizar las herramientas para generar esqueletos HTML para cada uno de los paneles que hay en el archivo PDML. En tiempo de ejecución, se visualizará el tema de ayuda correcto cuando el usuario pulse el botón Ayuda del panel o pulse la tecla F1 mientras el foco se encuentre en uno de los controles del panel. Debe insertar el contenido de la ayuda en los puntos adecuados del HTML, dentro del ámbito de los códigos <!-- `HELPDOC:SEGMENTBEGIN` --> y <!-- `HELPDOC:SEGMENTEND` -->. Encontrará información de ayuda más específica en Edición de los documentos de ayuda generados por el Constructor de GUI.

Puede generar esqueletos de código fuente para los JavaBeans que van a suministrar los datos de un panel. Utilice la ventana Propiedades del Constructor de GUI para rellenar las propiedades DATACLASS y ATTRIBUTE de los controles que contendrán datos. La propiedad DATACLASS identifica el nombre de clase del bean y la propiedad ATTRIBUTE especifica el nombre de los métodos obtenedores/establecedores implementados por la clase bean. Tras añadir esta información al archivo PDML, puede utilizar el Constructor de GUI para generar los esqueletos de código fuente Java y compilarlos. En el momento de la ejecución, se llamará a los métodos obtenedores/establecedores adecuados para rellenar los datos del panel.

Nota: el número y el tipo de los métodos obtenedores/establecedores varían en función del tipo de control de interfaz de usuario al que estén asociados los métodos. Los protocolos de los métodos para cada control están documentados en la descripción de la clase DataBean.

Por último, puede serializar el contenido del archivo PDML. La serialización genera una representación binaria compacta de todos los recursos de interfaz de usuario del archivo. Esto supone una notable mejora en el rendimiento de la interfaz de usuario, dado que no es preciso interpretar el archivo PDML para visualizar los paneles.

Resumiendo: si ha creado un archivo PDML denominado **MisPaneles.pdml**, también se generarán los archivos siguientes, en función de las opciones que haya seleccionado en las herramientas:

- **MisPaneles.properties** si ha definido el paquete de recursos como archivo PROPERTIES.
- **MisPaneles.java** y **MisPaneles.class** si ha definido el paquete de recursos como subclase

ListResourceBundle.

- **<nombre panel>.html** para cada uno de los paneles del archivo PDML, si ha elegido generar esqueletos de ayuda en línea.
- **<nombre clase datos>.java** y **<nombre clase datos>.class** para cada una de las clases bean exclusivas que haya especificado en las propiedades de DATACLASS, si ha elegido generar esqueletos de código fuente para los JavaBeans.
- **<nombre recurso>.pdml.ser** para cada uno de los recursos de interfaz de usuario definidos en el archivo PDML, si ha elegido serializar su contenido.

Nota: las funciones de comportamiento condicional (SELECTED/DESELECTED) no funcionarán si el nombre del panel coincide con el del panel al que se vaya a conectar la función de comportamiento condicional. Por ejemplo, si el PANEL1 del ARCHIVO1 tiene una referencia a comportamiento condicional conectada a un campo que hace referencia a un campo del PANEL1 del ARCHIVO2, el evento de comportamiento condicional no funcionará. Para arreglarlo, basta con cambiar el nombre del PANEL1 del ARCHIVO2 y después actualizar el evento de comportamiento condicional en el ARCHIVO1 para que refleje el cambio realizado.

Ejecución del convertidor de scripts de recursos

Para iniciar el convertidor de scripts de recursos, invoque el intérprete de Java con el mandato siguiente:

```
java com.ibm.as400.ui.tools.PDMLViewer
```

Si no ha establecido la variable de entorno CLASSPATH de modo que contenga los archivos JAR de la Caja de Herramientas Gráfica, tendrá que especificarlos en la línea de mandatos mediante la opción `classpath`. Consulte el apartado Puesta a punto de la Caja de Herramientas Gráfica.

También puede ejecutar el convertidor de scripts de recursos en modalidad de proceso por lotes mediante este mandato:

```
java com.ibm.as400.ui.tools.RC2XML archivo [opciones]
```

donde *archivo* es el nombre del archivo RC (script de recursos) que se ha de procesar. **Opciones**

`-x` *nombre*

El nombre del archivo PDML generado. Toma por omisión el nombre del archivo RC que se ha de procesar.

`-p` *nombre*

El nombre del archivo PROPERTIES generado. Toma por omisión el nombre del archivo PDML.

`-r` *nombre*

El nombre de la subclase ListResourceBundle generada. Toma por omisión el nombre del archivo PDML.

`-package` *nombre*

El nombre del paquete al que se asignarán los recursos generados. De no especificarse, no se generará ninguna sentencia de paquete.

`-l` *entorno_nacional*

El entorno nacional en el que se deben generar los recursos generados. Si se especifica un entorno nacional, se añadirán como sufijos al nombre del paquete de recursos generados los códigos ISO de dos caracteres correspondientes al idioma y al país o región.

`-h`

Genera esqueletos HTML para la ayuda en línea.

`-d`

Genera esqueletos de código fuente para JavaBeans.

`-s`

Serializa todos los recursos.

Correlación entre los recursos de Windows y PDML

Todos los diálogos, menús y tablas de series que se encuentren en el archivo RC se convertirán a los correspondientes recursos de la Caja de Herramientas Gráfica del archivo PDML generado. También puede definir las propiedades DATACLASS y ATTRIBUTE para los controles de Windows que se propagarán al nuevo archivo PDML siguiendo un simple convenio de denominación cuando cree los identificadores para los recursos de Windows. Estas propiedades se utilizarán para generar esqueletos de código fuente para los JavaBeans cuando ejecute la conversión.

El convenio de denominación para los identificadores de recursos de Windows es el siguiente:

`IDCB_<nombre clase>_<atributo>`

donde `<nombre clase>` es el nombre totalmente calificado de la clase bean que desea designar como propiedad DATACLASS del control y `<atributo>` es el nombre de la propiedad de bean que desea designar como propiedad ATTRIBUTE del control.

Por ejemplo, un campo de texto de Windows con el ID de recurso `IDCB_com_MyCompany_MyPackage_MyBean_SampleAttribute` generaría una propiedad DATACLASS **com.MyCompany.MyPackage.MyBean** y una propiedad ATTRIBUTE **SampleAttribute**. Si elige generar JavaBeans al ejecutar la conversión, se generará el archivo fuente Java **MyBean.java**, que contendrá la sentencia de paquete **package com.MyCompany.MyPackage** y los métodos obtenedores y establecedores de la propiedad **SampleAttribute**.

Visualizar los paneles en tiempo de ejecución

La Caja de Herramientas Gráfica proporciona una API redistribuible que los programas Java pueden utilizar para visualizar paneles de interfaz de usuario definidos mediante PDML. Para visualizar los paneles, la API interpreta el lenguaje PDML y representa la interfaz de usuario utilizando las clases JFC (clases Java fundamentales).

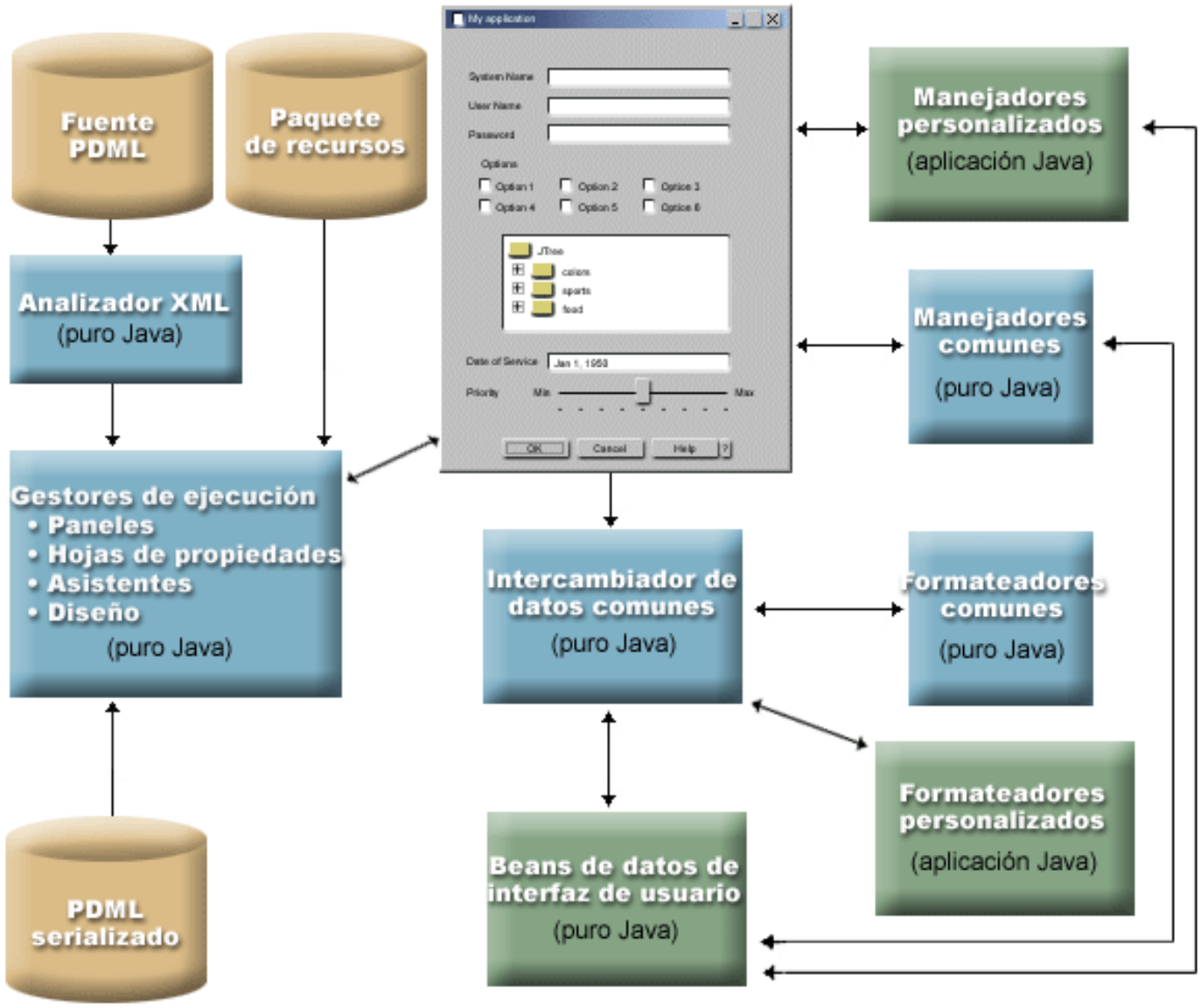
El entorno de ejecución de la Caja de Herramientas Gráfica proporciona los servicios siguientes:

- Maneja todos los intercambios de datos entre los controles de interfaz de usuario y los JavaBeans que se hayan identificado en el PDML.
- Lleva a cabo la validación de los datos de usuario para los tipos de datos de carácter y entero y define una interfaz que permite implementar una validación personalizada. Si los datos no son válidos, se muestra al usuario un mensaje de error.
- Define un proceso estandarizado para los eventos de Comprometer, Cancelar y Ayuda, y proporciona una infraestructura para manejar eventos personalizados.
- Gestiona las interacciones entre los controles de la interfaz de usuario basándose en la información de estado definida en el PDML. (Por ejemplo, podría ser conveniente inhabilitar un grupo de controles cada vez que el usuario seleccionase un determinado botón de selección.)

El paquete com.ibm.as400.ui.framework.java contiene la API de la unidad de ejecución Caja de Herramientas Gráfica.

Los elementos del entorno de ejecución Caja de Herramientas Gráfica se muestran en la figura 1. El programa Java es un cliente de uno o más objetos del recuadro **Gestores de ejecución**.

Figura 1: entorno de ejecución Caja de Herramientas Gráfica



Ejemplos

Supongamos que el panel **MyPanel** se ha definido en el archivo **TestPanels.pdml**, y que se ha asociado un archivo de propiedades **TestPanels.properties** a la definición del panel. Los dos archivos residen en el directorio **com/ourCompany/ourPackage**, al que se puede acceder desde un directorio definido en la vía de acceso de clases o desde un archivo ZIP o JAR definido en la vía de acceso de clases.

Ejemplo: crear y visualizar un panel

El código siguiente crea y visualiza el panel:

```
import com.ibm.as400.ui.framework.java.*;

// Cree el gestor de paneles. Parámetros:
// 1. Nombre de recurso de la definición de panel
// 2. Nombre del panel
// 3. Lista de DataBeans omitidos

PanelManager pm = null;
try {
  pm = new PanelManager("com.ourCompany.ourPackage.TestPanels",
                        "MyPanel", null);
}

catch (DisplayManagerException e) {
            e.displayUserMessage(null);
      System.exit(-1);
}

// Visualice el panel
pm.setVisible(true);
```

Ejemplo: crear un diálogo

Una vez implementados los DataBeans que suministran datos al panel e identificados los atributos en el PDML, puede utilizarse el código siguiente para construir un diálogo totalmente operativo:

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

// Cree una instancia de los objetos que suministran datos al panel
TestDataBean1 db1 = new TestDataBean1();
TestDataBean2 db2 = new TestDataBean2();

// Inicialice los objetos
db1.load();
db2.load();

// Configure para pasar los objetos a la infraestructura de UI
DataBean[] dataBeans = { db1, db2 };

// Cree el gestor de paneles. Parámetros:
// 1. Nombre de recurso de la definición de panel
// 2. Nombre del panel
// 3. Lista de DataBeans
// 4. Ventana del marco propietario

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);
}

// Visualice el panel
pm.setVisible(true);
```

Ejemplo: cómo se utiliza el gestor de paneles dinámico

Hemos añadido un nuevo servicio al gestor de paneles existente. Se trata del gestor de paneles dinámico, que dimensiona dinámicamente el panel en tiempo de ejecución. Volvamos al ejemplo de **MyPanel**, ahora con el gestor de paneles dinámico:

```
import com.ibm.as400.ui.framework.java.*;

// Cree el gestor de paneles dinámico. Parámetros:
// 1. Nombre de recurso de la definición de panel
// 2. Nombre del panel
// 3. Lista de DataBeans omitidos

DynamicPanelManager dpm = null;
try {
  pm = new DynamicPanelManager("com.ourCompany.ourPackage.TestPanels",
                               "MyPanel", null);
}

catch (DisplayManagerException e) {
            e.displayUserMessage(null);
      System.exit(-1);
}

// Visualice el panel
pm.setVisible(true);
```

Al crear una instancia de esta aplicación de panel, podrá ver la característica de dimensionamiento dinámico de los paneles. Sitúe el cursor en el borde de la pantalla de la GUI y, cuando vea las flechas de dimensionamiento, podrá cambiar el tamaño del panel.

Edición de los documentos de ayuda generados por el Constructor de GUI

Para cada archivo de proyecto PDML, el Constructor de GUI genera un esqueleto de ayuda y lo pone en un documento HTML individual. Antes de utilizarse, este archivo HTML se desglosa en archivos HTML de un solo tema para cada diálogo del proyecto HTML. Así se proporciona al usuario ayuda granular para cada tema y además usted solo tendrá que gestionar unos pocos archivos de ayuda de gran tamaño.

El documento de ayuda es un archivo HTML válido que se puede ver en cualquier navegador y editar mediante la mayoría de los editores de HTML. Los códigos que definen las secciones de un documento de ayuda están incorporados en los comentarios, por lo que no se muestran en un navegador. Los códigos de comentario permiten desglosar el documento de ayuda en varias secciones:

- Cabecera
- Sección de los temas de cada diálogo
- Sección de los temas de cada control habilitado para la ayuda
- Pie de página

También puede añadir más secciones de temas antes del pie de página para facilitar información adicional o de carácter general. 2Las secciones dedicadas a los temas solo tienen el cuerpo html hasta que se subdividen, momento en el que se crea una cabecera y un pie de página. Cuando se desglosa el documento de ayuda, el procesador añade una cabecera y un pie de página a la sección del tema para hacer un archivo HTML completo. Como cabecera y pie de página por omisión se toma la cabecera y el pie de página del documento de ayuda. Sin embargo, usted puede alterar temporalmente la cabecera por omisión con la suya propia.

Dentro del documento de ayuda

Las secciones que explican las partes de que consta el documento de ayuda son las siguientes:

Cabecera

Al final de la sección de cabecera hay este código:
<!-- HELPDOC:HEADEREND -->
Si desea alterar temporalmente la cabecera por omisión de todos los temas individuales cuando se subdividen, utilice la palabra clave HEADER y proporcione el nombre de un fragmento html que deba incluirse. Por ejemplo:
<!-- HELPDOC:HEADEREND HEADER="defaultheader.html" -->

Segmento de tema

Cada tema se pone entre estos códigos:
<!-- HELPDOC:SEGMENTBEGIN --> y <!-- HELPDOC:SEGMENTEND -->
Inmediatamente a continuación del código SEGMENTBEGIN hay un código de ancla que nombra el segmento. También proporciona el nombre de archivo del documento HTML que se crea cuando se subdivide el documento de ayuda. El nombre del segmento es una combinación formada por el identificador del panel, el identificador del control y la extensión del archivo futuro (html). Por ejemplo: "MY_PANEL.MY_CONTROL.html". Los segmentos de los paneles sólo tienen el identificador del panel y la extensión del archivo futuro.

El generador de ayuda colocará en el documento texto que indique dónde debe colocarse la información de ayuda:

```
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" --><A
NAME="MY_PANEL.MY_CONTROL.html"></A>
<H2>Mi control favorito</H2>
Inserte aquí la ayuda de "Mi control favorito".
<P><!-- HELPDOC:SEGMENTEND -->
```

Puede añadir códigos HTML 2.0 adicionales después del código de ancla y antes del código SEGMENTEND.

El código PDMLSYNCH controla hasta qué punto está ligado un segmento a los controles definidos en PDML. Si PDMLSYCH es "YES", el segmento del documento de ayuda se eliminará si se elimina en el PDML el control que tiene el mismo nombre. PDMLSYNCH="NO" indica que el tema debe conservarse en el documento de ayuda sin tener en cuenta si existe un control correspondiente en el PDML. Esto se utiliza, por ejemplo, cuando se crean temas adicionales para profundizar o cuando se crea un tema común.

La ayuda generada para un panel tiene enlaces con cada control habilitado para la ayuda y existente en el panel. Estos enlaces se generan con una referencia de ancla local, por lo que puede probarlos como enlaces internos en un navegador estándar. Cuando se subdivide el documento de ayuda, el procesador elimina el "#" que hay en esos enlaces internos convirtiéndolos así en enlaces externos de los archivos HTML de tema individual resultantes. Debido a que puede ser conveniente tener enlaces internos dentro de un tema, el procesador solo elimina los "#" precedentes cuando la referencia tiene incorporada en ella la extensión ".html".

Si desea alterar temporalmente la cabecera por omisión de un determinado tema, utilice la palabra clave HEADER y proporcione el nombre de un fragmento html que deba incluirse. Por ejemplo:
`<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" HEADER="specialheader.html" -->`

Pie de página

El pie de página de un documento de ayuda empieza por este código:
`<!-- HELPDOC:FOOTERBEGIN -->`.El pie de página estándar es </BODY></HTML>. Este pie de página se añade a cada archivo HTML.

Adición de enlaces

Puede añadir enlaces con cualquier URL externo o interno, así como también con cualquier otro segmento. Sin embargo, debe ajustarse a algunos convenios:

- Los URL externos se utilizan de manera estándar. Ello incluye los enlaces internos con los URL externos.
- Los enlaces internos dentro del mismo tema se escriben de forma estándar, pero la extensión ".html" no debe formar parte del nombre del código. Ello se debe a que el procesador del documento de ayuda supone que los enlaces que tengan la extensión .html tendrán que ser externos cuando los temas se separen. Por lo tanto, elimina el "#" precedente.
- Los enlaces con los otros segmentos del tema se deben escribir con un "#" precedente como si fuesen una referencia de ancla interna.
- Los enlaces internos con los otros segmentos del tema también se pueden crear. Solo se elimina el "#" inicial durante el proceso.

Notas:

- En tiempo de ejecución, la clase PanelManager busca los archivos de ayuda en un subdirectorio que tiene el mismo nombre que el archivo PDML. El procesador, cuando subdivide el documento de ayuda, crea por omisión este subdirectorio y coloca en él los archivos HTML resultantes.
- El procesador no hace ningún ajuste para las referencias de URL externos que sean enlaces relativos. Cuando se enlaza desde un archivo de tema individual, los enlaces relativos harán la búsqueda desde el nuevo subdirectorio. Por lo tanto, tendrá que colocar copias de los recursos (por ejemplo, imágenes) allí donde puedan encontrarse o bien utilizar "../" en la vía de acceso para realizar la búsqueda desde el directorio del panel.

Edición mediante un editor visual

Puede editar el contenido de la ayuda en casi todos los editores HTML visuales. Debido a que los códigos HELPDOC son comentarios, tal vez no sean evidentes en algunos editores. Por comodidad, se añade al esqueleto de la ayuda una raya horizontal situada inmediatamente antes del código SEGMENTBEGIN y otra inmediatamente después del código SEGMENTEND. Estas rayas horizontales proporcionan una indicación visual clara de todo el segmento en un editor visual. Si selecciona un segmento porque desea moverlo, copiarlo o suprimirlo, seleccione las rayas horizontales que lo envuelven para asegurarse de que ha incluido los códigos SEGMENTBEGIN y SEGMENTEND

en su selección. Estas rayas horizontales no se copian en los archivos HTML individuales finales.

Creación de temas adicionales

Puede crear segmentos de temas adicionales en el documento de ayuda. Para ello, lo más sencillo suele ser copiar otro segmento. Cuando lo haga, debe copiar asimismo las rayas horizontales que hay justo antes del código SEGMENTBEGIN y justo después del código SEGMENTEND. Así la edición visual futura será mucho más fácil y usted se asegura de que no falte ningún código de la pareja. Para obtener los mejores resultados, siga estos consejos:

- El nombre del ancla debe ser el nombre que desea dar al archivo individual resultante cuando se subdivida el documento de ayuda. Debe acabar en ".html".
- Debe utilizar la palabra clave PDMLSYNCH="NO" en el código SEGMENTBEGIN para impedir que el segmento se elimine si se vuelve a generar el esqueleto de la ayuda.
- Las referencias al nuevo tema se harán como enlace interno del documento de ayuda con un "#" precedente. Este "#" se eliminará en el proceso posterior cuando los segmentos se subdividan en archivos individuales.

Comprobación de los enlaces

En la mayoría de los casos de escritura, puede comprobar sus enlaces examinando el documento en un navegador Web y seleccionando los distintos enlaces. En el documento de ayuda individual, los enlaces aún tienen el formato interno.

Cuando haya terminado o cuando desee hacer la prueba con la aplicación para la que está desarrollando la ayuda, tendrá que desglosar el documento de ayuda en archivos individuales. Para ello, utilice el Proceso de documento de ayuda a HTML.

Si necesita volver a generar el documento de ayuda después de editarlo, su escrito se conservará. Puede ser conveniente volver a generar el documento de ayuda si se añaden nuevos controles una vez generado el esqueleto de ayuda original. En tal caso, el generador de ayuda comprueba si hay un documento de ayuda existente antes de crear un esqueleto nuevo. Si lo encuentra, conserva los segmentos existentes y luego añade los nuevos controles.

Caja de Herramientas Gráfica - Ejemplos

Hemos proporcionado ejemplos para mostrarle cómo implementar para sus propios programas de UI las herramientas dentro de la Caja de Herramientas Gráfica.

- Construir y visualizar un panel: muestra cómo se construye un simple panel. Después, el ejemplo enseña cómo se construye una pequeña aplicación Java que visualice el panel. Cuando el usuario entre datos en el campo de texto y pulse el botón Cerrar, la aplicación devolverá los datos a la consola Java. Este ejemplo ilustra las características y el funcionamiento básicos del entorno de la Caja de Herramientas Gráfica como conjunto.
- Crear y visualizar un panel: muestra cómo se crea y se visualiza un panel en el caso particular de que el panel se encuentre en el mismo directorio que el archivo de propiedades.
- Construir un diálogo totalmente operativo: una vez implementados los DataBeans que suministran datos al panel e identificados los atributos en el PDML, este ejemplo muestra cómo se construye un diálogo totalmente operativo.
- Ajustar el tamaño de un panel utilizando el gestor de paneles dinámico: el gestor de paneles dinámico permite dimensionar dinámicamente el panel en tiempo de ejecución.
- Cuadro combinado editable: muestra un ejemplo de codificación de beans de datos para un cuadro combinado editable.

Los ejemplos siguientes muestran cómo puede ayudarle el Constructor de GUI a crear:

- Paneles: muestra cómo se crea un panel de ejemplo y el código de beans de datos que ejecuta el panel
- Secciones baraja: muestra cómo se crea una sección baraja y qué aspecto tendría una sección baraja acabada
- Hojas de propiedades: muestra cómo se crea una hoja de propiedades y qué aspecto tendría una hoja de propiedades acabada
- Secciones divididas: muestra cómo se crea una sección dividida y qué aspecto tendría una sección dividida acabada
- Secciones con pestañas: muestra cómo se crea una sección con pestañas y qué aspecto tendría una sección con pestañas acabada
- Asistentes: muestra cómo se crea un asistente y qué aspecto tendría el producto acabado
- Barras de herramientas: muestra cómo se crea una barra de herramientas y qué aspecto tendría una barra de herramientas acabada
- Barras de menús: muestra cómo se crea una barra de menús y qué aspecto tendría una barra de menús acabada
- Ayuda: muestra cómo se genera un documento de ayuda y los procedimientos que permiten desglosarlo en páginas temáticas. Consulte asimismo Edición de los documentos de ayuda generados por el Constructor de GUI
- Ejemplo: muestra el aspecto que puede tener un programa PDML completo, con paneles, una hoja de propiedades, un asistente, opciones para seleccionar/deseleccionar y opciones de menú.

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Todos los programas contenidos aquí se proporcionan "TAL CUAL" sin garantías de ningún tipo. Las garantías implícitas de no incumplimiento, comerciabilidad y adecuación para un fin determinado se especifican explícitamente como declaraciones de limitación de responsabilidad.

Utilización de la Caja de Herramientas Gráfica en un navegador

La Caja de Herramientas Gráfica permite construir paneles para applets Java que se ejecuten en un navegador Web. Este apartado describe el procedimiento que debe seguirse para convertir el simple panel del ejemplo de la Caja de Herramientas Gráfica de modo que pueda ejecutarse en un navegador. Los niveles mínimos de navegador soportados son Netscape 4.05 e Internet Explorer 4.0. Para que no tenga que preocuparse de las particularidades de cada navegador, le recomendamos que utilice Java Plug-in de Sun para ejecutar los applets. De lo contrario, tendría que construir unos archivos JAR firmados para Netscape y otros archivos CAB firmados para Internet Explorer.

Construcción del applet

El código que permite visualizar un panel en un applet es prácticamente idéntico al código que se utiliza en el ejemplo de la aplicación Java; pero primero es preciso empaquetar de nuevo el código en el método **`init`** de una subclase **JApplet**. Asimismo, debemos añadir algún fragmento de código para asegurarnos de que el tamaño del panel del applet se ajuste a las dimensiones especificadas en la definición PDML del panel. A continuación figura el código fuente para nuestro applet de ejemplo, llamado **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
{
    // Estas líneas de código son necesarias para mantener el tamaño del panel.
    private PanelManager        m_pm;
    private Dimension           m_panelSize;

    // Defina una excepción que deba lanzarse en el caso de que se produzca algún error.
    class SampleAppletException extends RuntimeException {}

    public void init()
    {
        System.out.println("Se está inicializando");

        // Rastree los parámetros del applet.
        System.out.println("Base de código de SampleApplet=" + getCodeBase());
        System.out.println("Base de documento de SampleApplet=" + getDocumentBase());

        // Compruebe que se está ejecutando una máquina virtual Java compatible con Swing 1.1.
        if (System.getProperty("java.version").compareTo("1.1.5") < 0)
            throw new IllegalStateException("SampleApplet no puede ejecutarse en una VM Java cuya
versión sea " +
                                            System.getProperty("java.version") + " - requiere
1.1.5 o superior");

        // Cree una instancia del objeto bean que suministra datos al panel.
        SampleBean bean = new SampleBean();

        // Inicialice el objeto.
        bean.load();

        // Configure para pasar el bean al gestor de paneles.
```

```
    DataBean[] beans = { bean };

    // Actualice la barra de estado.
    showStatus("Se está cargando la definición de panel...");

    // Cree el gestor de paneles. Parámetros:
    // 1. Archivo PDML como nombre de recurso
    // 2. Nombre del panel que se ha de visualizar
    // 3. Lista de objetos de datos que suministran los datos del panel
    // 4. La sección de contenido del applet

    try { m_pm = new PanelManager("MyGUI", "PANEL_1", beans, getContentPane()); }
    catch (DisplayManagerException e)
    {
        // Algo no ha funcionado; se ha de visualizar un mensaje y salir.
        e.displayUserMessage(null);
        throw new SampleAppletException();
    }

    // Identifique el directorio en el que reside la ayuda en línea.
    m_pm.setHelpPath("http://MyDomain/MyDirectory/");

    // Visualice el panel
    m_pm.setVisible(true);
}

public void start()
{
    System.out.println("Se está iniciando");

    // Ajuste el tamaño del panel al tamaño predefinido.
    m_panelSize = m_pm.getPreferredSize();
    if (m_panelSize != null)
    {
        System.out.println("Se está ajustando el tamaño a " + m_panelSize);
        resize(m_panelSize);
    }
      else
        System.err.println("Error: getPreferredSize ha devuelto null");
}

public void stop()
{
    System.out.println("Se está deteniendo");
}

public void destroy()
{
    System.out.println("Se está destruyendo");
}

public void paint(Graphics g)
{
```

```
        // Llame al padre en primer lugar.
        super.paint(g);

        // Conserve el tamaño predefinido del panel al redibujar (repaint).
        if (m_panelSize != null)
            resize(m_panelSize);
    }
}
```

La sección de contenido del applet se pasa a la Caja de Herramientas Gráfica en forma del contenedor que se ha de diseñar. En el método **start**, ajustamos el tamaño de la sección del applet al tamaño correcto y alteramos temporalmente el método **paint** (dibujar) para conservar el tamaño del panel cuando se redimensione la ventana del navegador.

Al ejecutar la Caja de Herramientas Gráfica en un navegador, no se puede acceder a los archivos HTML de la ayuda en línea del panel desde un archivo JAR. Estos archivos deben residir como archivos separados en el directorio en el que reside el applet. La llamada a **`PanelManager.setHelpPath`** identifica este directorio para la Caja de Herramientas Gráfica, para que se puedan localizar los archivos de ayuda.

Códigos HTML

Debido a que recomendamos que se utilice Java Plug-in de Sun para proporcionar el nivel correcto del entorno de ejecución Java, el lenguaje HTML que permite identificar un applet de la Caja de Herramientas Gráfica no es tan sencillo como sería de desear. Afortunadamente, es posible reutilizar para otros applets una misma plantilla HTML, tan solo con hacer pequeños cambios. Los códigos están diseñados para que puedan interpretarse tanto en Netscape Navigator como en Internet Explorer, y generan una solicitud para bajar Java Plug-in del sitio Web de Sun si todavía no se ha instalado en la máquina del usuario. Encontrará información detallada sobre cómo funciona Java Plug-in en Especificación HTML de Java Plug-in.

A continuación figura nuestro código HTML para el applet de ejemplo, en el archivo **MyGUI.html**:

```
<html>

<head>
<title>Demostración de la Caja de Herramientas Gráfica</title>
</head>

<body>
<h1>Demostración de la Caja de Herramientas Gráfica utilizando Java(TM) Plug-in</h1>
<p>

<!-- PRINCIPIO DE LOS CÓDIGOS DE APPLET DE JAVA(TM) PLUG-IN -->

<!-- Los códigos siguientes utilizan una sintaxis especial que permite que tanto Netscape como
Internet Explorer carguen   -->
<!-- Java Plug-in y ejecuten el applet en el JRE del conector (plug-in). No modifique esta
sintaxis.              -->
<!-- Encontrará más información en
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 se ha encontrado soporte para applets de JDK 1.1!
        </NOEMBED>
    </EMBED>
</OBJECT>

<!-- FIN DE LOS CÓDIGOS DE APPLET DE JAVA(TM) PLUG-IN -->

<p>
</body>
</html>
```

Es importante que la información de versión se establezca para 1.1.3.

Nota: en este ejemplo, hemos elegido almacenar el archivo JAR del analizador XML, **x4j400.jar**, en el servidor Web. Ello sólo es necesario cuando se incluye el archivo PDML como parte de la instalación del applet. Por motivos de rendimiento, lo normal es *serializar* las definiciones de panel para que la Caja de Herramientas Gráfica no tenga que interpretar el archivo PDML en tiempo de ejecución. Esto supone una notable mejora en el rendimiento de la interfaz de usuario al crear representaciones binarias compactas de los paneles. Si desea obtener más información al respecto, consulte la descripción de los archivos generados por las herramientas.

Instalación y ejecución del applet

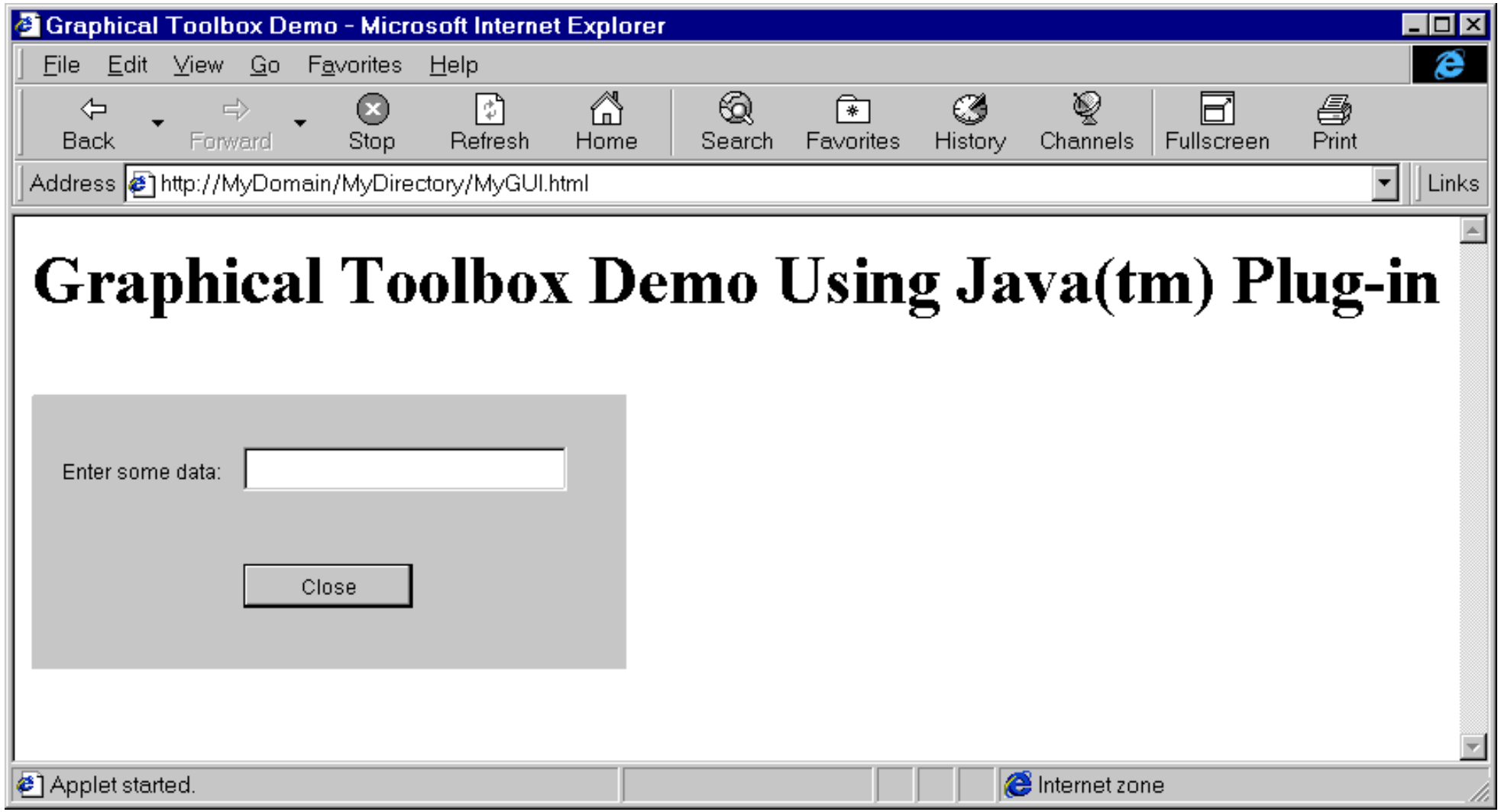
Para instalar el applet en su servidor Web preferido, siga estos pasos:

- Compile **SampleApplet.java**.
- Cree un archivo JAR llamado **MyGUI.jar** que ha de contener los archivos binarios del applet. Estos archivos son los archivos de clase generados al compilar **SampleApplet.java** y **SampleBean.java**, el archivo PDML **MyGUI.pdml** y el paquete de recursos **MyGUI.properties**.
- Copie el nuevo archivo JAR en el directorio que desee del servidor Web. Copie en el directorio del servidor los archivos HTML que contienen la ayuda en línea.
- Copie en el directorio del servidor los archivos JAR de la Caja de Herramientas Gráfica.
- Por último, copie en el directorio del servidor el archivo HTML **MyGUI.html** que contiene el applet incorporado.

Consejo: cuando pruebe los applets, no olvide eliminar los archivos JAR de la Caja de Herramientas Gráfica de la variable de entorno CLASSPATH de la estación de trabajo. De lo contrario, recibiría mensajes de error que le indicarían que no se pueden localizar los recursos del applet en el servidor.

Ahora ya está listo para ejecutar el applet. Vaya al archivo **MyGUI.html** del servidor con el navegador Web. Si todavía no tiene instalado Java Plug-in, se le preguntará si desea instalarlo. Tras instalar el conector e iniciar el applet, la pantalla del navegador será parecida a la que muestra la figura 1:

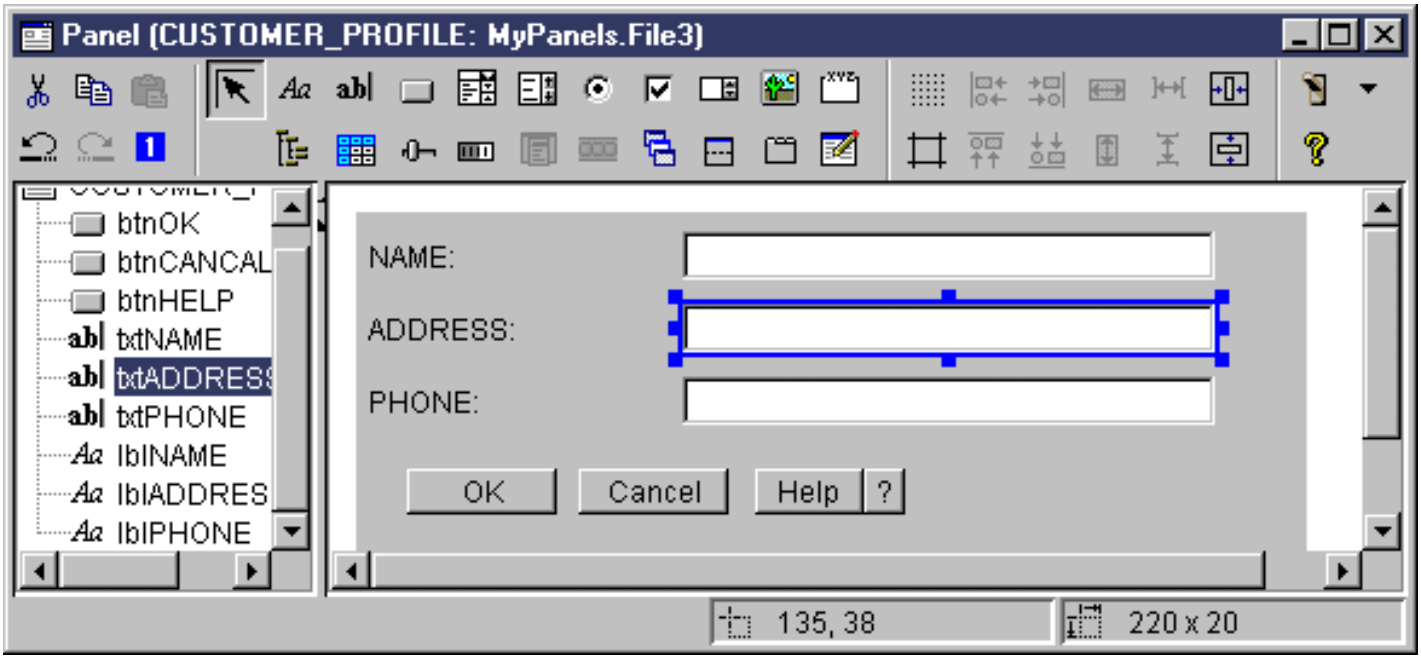
Figura 1: ejecución del applet de ejemplo en un navegador


Graphical Toolbox Demo - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Print
Address http://MyDomain/MyDirectory/MyGUI.html
Links
Graphical Toolbox Demo Using Java(tm) Plug-in
Enter some data:
Close
Applet started.
Internet zone

Barra de herramientas del constructor de paneles del Constructor de GUI

La figura 1 muestra la ventana Constructor de paneles del Constructor de GUI. A continuación de la figura 1 se encuentra una lista en la que consta cada icono de herramienta del constructor de paneles y se describe su función.

Figura 1: ventana Panel del Constructor de GUI



Pulse Puntero para mover un componente en un panel y ajustar su tamaño.

Pulse Etiqueta para insertar una etiqueta estática en un panel.

Pulse Texto para insertar un cuadro de texto en un panel.

Pulse Botón para insertar un botón en un panel.

Pulse Cuadro combinado para insertar un cuadro de lista desplegable en un panel.

Pulse Cuadro de lista para insertar un cuadro de lista en un panel.

Pulse Botón de selección para insertar un botón de selección en un panel.

Pulse Recuadro de selección para insertar un recuadro de selección en un panel.

Pulse Selector cíclico para insertar un selector cíclico en un panel.

Pulse Imagen para insertar una imagen en un panel.

Pulse Barra de menús para insertar una barra de menús en un panel.

Pulse Cuadro de grupo para insertar un cuadro de grupo con etiquetas en un panel.

Pulse Árbol para insertar un árbol jerárquico en un panel.

 Pulse Tabla para insertar una tabla en un panel.

 Pulse Graduador para insertar un graduador ajustable en un panel.

 Pulse Barra de progreso para insertar una barra de progreso en un panel.

 Pulse Sección baraja para insertar una sección baraja en un panel.Una sección baraja consta de una pila de paneles. El usuario puede seleccionar cualquiera de los paneles, pero sólo el panel seleccionado está plenamente visible.

 Pulse Sección dividida para insertar una sección dividida en un panel.Una sección dividida consta de dos secciones horizontales o verticales.

 Pulse Sección con pestañas para insertar una sección con pestañas en un panel.Una sección con pestañas consta de un conjunto de paneles con pestañas en la parte superior. El usuario pulsa una pestaña para visualizar el contenido de un panel. El título del panel se utiliza como texto de una pestaña.

 Pulse Personalizado para insertar en un panel un componente de interfaz de usuario definido de modo personalizado.

 Pulse Barra de herramientas para insertar una barra de herramientas en un panel.

 Pulse Conmutador de cuadrícula para habilitar una cuadrícula en un panel.

 Pulse Alinear superior para alinear varios componentes de un panel con el borde superior de un componente específico o primario.

 Pulse Alinear inferior para alinear varios componentes de un panel con el borde inferior de un componente específico o primario.

 Pulse Igualar altura para igualar la altura de varios componentes con la altura de un componente específico o primario.

 Pulse Centrar verticalmente para centrar verticalmente un componente seleccionado en relación con el panel.

 Pulse Conmutar márgenes para ver los márgenes del panel.

 Pulse Alinear izquierda para alinear varios componentes de un panel con el borde izquierdo de un componente específico o primario.

 Pulse Alinear derecha para alinear varios componentes de un panel con el borde derecho de un componente específico o primario.

 Pulse Igualar anchura para igualar la anchura de varios componentes con la anchura de un componente específico o primario.

 Pulse Centrar horizontalmente para centrar horizontalmente un componente seleccionado en relación con el panel.

 Pulse Cortar para cortar componentes del panel.

 Pulse el botón Copiar para copiar componentes de panel.

 Pulse Pegar para pegar componentes de panel entre distintos paneles o archivos.

 Pulse Deshacer para deshacer la última acción.

 Pulse Rehacer para rehacer la última acción.

 Pulse Orden de tabulador para controlar el orden de selección de cada componente del panel cuando el usuario pulsa la tecla de tabulador para navegar por el panel.

 Pulse Vista previa para ver una vista previa del aspecto de un panel.

 Pulse Ayuda para obtener información más específica acerca de la Caja de Herramientas Gráfica.

Beans de IBM Toolbox para Java

Los JavaBeans(TM) son componentes de software reutilizables que están escritos en Java. El componente es un fragmento de código de programa que proporciona una unidad funcional bien definida; puede ser tan pequeño como una etiqueta para un botón de una ventana o tan grande como toda una aplicación.

Los JavaBeans pueden ser componentes visuales o no visuales. Aun así, los JavaBeans no visuales disponen de una representación visual (por ejemplo, un icono o un nombre) que permite una manipulación visual.

Todas las clases públicas de IBM Toolbox para Java también son JavaBeans. Estas clases se construyeron según los estándares JavaBean de Javasoft; funcionan como componentes reutilizables. Las propiedades y los métodos correspondientes a un bean Java de IBM Toolbox para Java son iguales a las propiedades y los métodos de la clase.

Los JavaBeans pueden emplearse dentro de un programa de aplicación o pueden manipularse visualmente en las herramientas constructoras, como el producto IBM VisualAge para Java.

Ejemplos

- Ejemplo: el ejemplo de bean de IBM Toolbox para Java muestra una forma de utilizar los JavaBeans en un programa.
- Ejemplo: el ejemplo de código de constructor visual de beans muestra una forma de crear un programa a partir de JavaBeans mediante un constructor visual de beans como IBM VisualAge para Java.

PCML (Program Call Markup Language)

Visión general

PCML (Program Call Markup Language) es un lenguaje de códigos que permite llamar a programas del servidor escribiendo menos código Java. El lenguaje PCML se basa en el lenguaje XML (Extensible Markup Language), una sintaxis de códigos que se utiliza para describir los parámetros de entrada y salida para los programas del servidor. PCML le permite definir códigos que describen de forma completa los programas del servidor llamados por la aplicación Java. Encontrará más información sobre XML en la sección Consulta sobre XML.

Una de las enormes ventajas de PCML es que no se tiene que escribir tanto código. Normalmente, se necesita código adicional para conectar, recuperar y convertir datos entre un servidor y los objetos de IBM Toolbox para Java. No obstante, con el lenguaje PCML, las llamadas al servidor con las clases de IBM Toolbox para Java se manejan de forma automática. Los objetos de las clases PCML se generan a partir de los códigos PCML y ayudan a minimizar la cantidad de código que se necesita escribir para llamar a los programas del servidor desde una aplicación.

Requisitos de plataforma

Si bien el lenguaje PCML se ha diseñado para dar soporte a las llamadas a programa distribuidas que se realizan a objetos de programas del servidor desde una plataforma Java, también se puede utilizar el lenguaje PCML para efectuar llamadas a un programa del servidor desde dentro del entorno del servidor.

Temas relacionados para obtener más información

Consulte los temas siguientes sobre cómo se utiliza el lenguaje PCML:

- Llamar a programas con la ayuda de PCML
- Construir llamadas a programa con los códigos PCML
- Un ejemplo de PCML

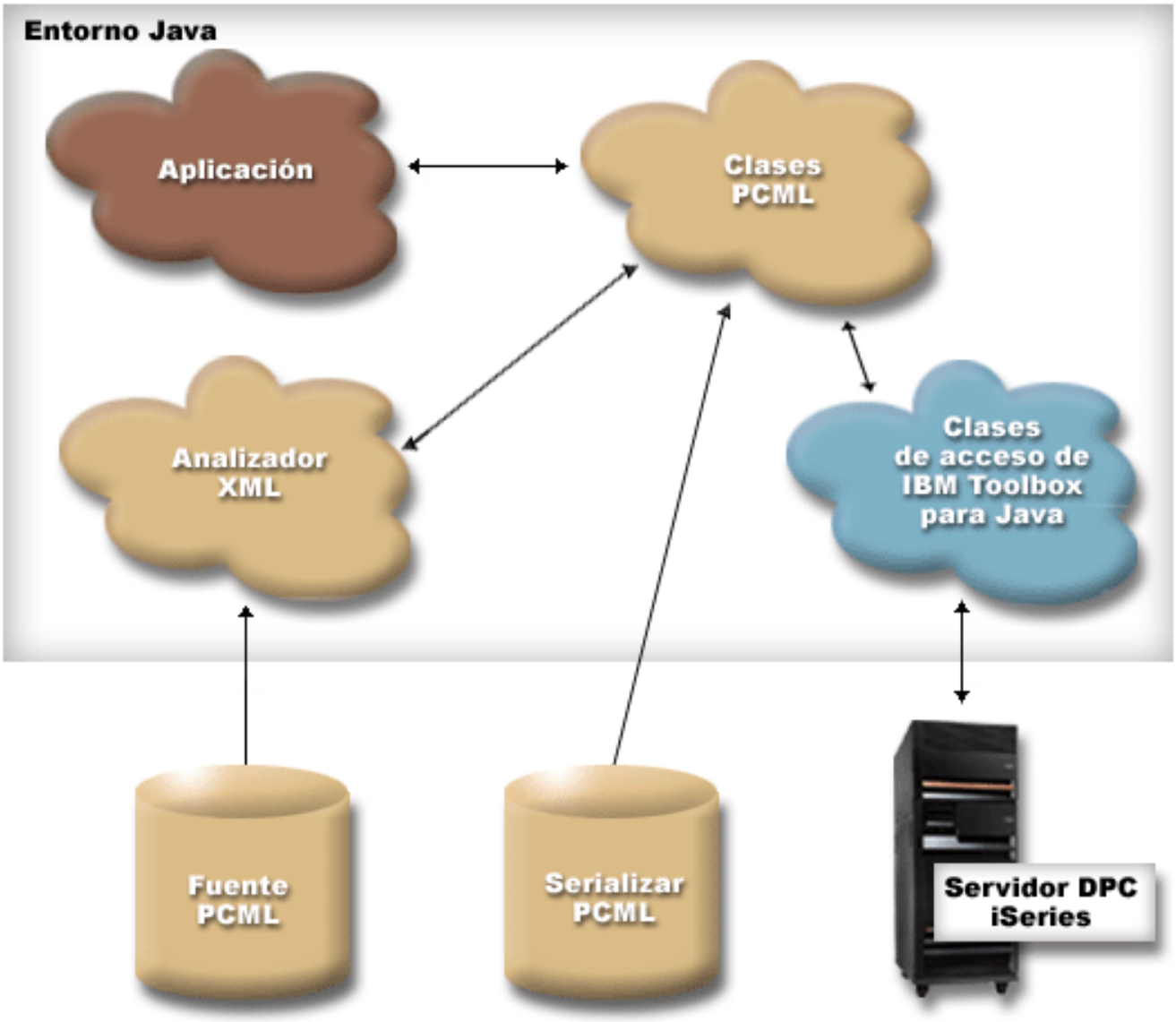
Construir llamadas a programa de iSeries con PCML

Para construir llamadas a programa de iSeries con PCML, debe empezar por crear una aplicación Java y un archivo fuente PCML.

En función del proceso de su diseño, debe escribir uno o varios archivos fuente PCML en los que describa las interfaces para los programas de iSeries a los que va a llamar su aplicación Java. Si desea consultar una descripción detallada de este lenguaje, vaya a Sintaxis de PCML.

A continuación, la aplicación Java interactúa con las clases PCML (en este caso, la clase ProgramCallDocument). La clase ProgramCallDocument utiliza el archivo fuente PCML para pasar información entre la aplicación Java y los programas de iSeries. La figura 1 muestra cómo interactúan las aplicaciones Java con las clases PCML.

Figura 1. Realización de llamadas a programa en el servidor mediante PCML.



Cuando la aplicación construye el objeto ProgramCallDocument, el analizador XML de IBM lee y analiza el archivo fuente PCML.

Una vez creada la clase ProgramCallDocument, el programa de aplicación utiliza los métodos de la clase ProgramCallDocument para recuperar la información necesaria del servidor mediante el servidor DPC (llamadas a programa distribuidas) de iSeries.

Para aumentar el rendimiento en tiempo de ejecución, puede serializarse la clase ProgramCallDocument durante el tiempo de construcción del producto. Luego, el documento de llamada a programa (ProgramCallDocument) se construye mediante el archivo serializado. En este caso, no se utiliza el analizador XML de IBM en tiempo de ejecución. Consulte el apartado Utilización de archivos PCML serializados.

Utilización de los archivos fuente PCML

La aplicación Java utiliza el lenguaje PCML construyendo un objeto ProgramCallDocument con una referencia al archivo fuente PCML. El objeto ProgramCallDocument trata el archivo fuente PCML como si fuese un recurso Java.

»La aplicación Java localiza el archivo fuente PCML mediante la CLASSPATH Java o el método setPath() de ProgramCallDocument. Utilice el método setPath() cuando el programa de la aplicación Java necesite establecer la vía de acceso del archivo PCML en tiempo de ejecución.«

El código Java siguiente construye un objeto ProgramCallDocument:

```
    AS400 as400 = new AS400();
    ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400,
"myPcmlDoc");
```

El objeto ProgramCallDocument buscará el fuente PCML en un archivo llamado myPcmlDoc.pcml. Observe que no se especifica la extensión .pcml en el constructor.

Si va a desarrollar una aplicación Java en un paquete Java, puede calificar por paquete el nombre del recurso PCML:

```
    AS400 as400 = new AS400();
    ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400,
"com.company.package.myPcmlDoc");
```

Utilización de los archivos PCML serializados

Para incrementar el rendimiento en tiempo de ejecución, puede utilizar un archivo PCML serializado. Un archivo PCML serializado contiene objetos Java serializados que representan el PCML. Los objetos serializados son los mismos objetos que se crean al construir el objeto ProgramCallDocument a partir de un archivo fuente tal como se ha descrito antes.

El hecho de utilizar archivos PCML serializados supone una mejora en el rendimiento porque no se necesita el analizador XML de IBM en tiempo de ejecución para procesar los códigos PCML.

El PCML se puede serializar mediante uno de estos procedimientos:

- Desde la línea de mandatos:

  ```
  java com.ibm.as400.data.ProgramCallDocument -serialize mypcml
  ```

  Este procedimiento resulta de gran utilidad para construir la aplicación mediante procesos por lotes.
- Desde un programa Java:

  ```
  ProgramCallDocument pcmlDoc; // Inicializado en algún otro lugar.
  pcmlDoc.serialize();
  ```

Si el código PCML se encuentra en un archivo fuente denominado myDoc.pcml, el resultado de la serialización es un archivo denominado myDoc.pcml.ser.

Los archivos fuente PCML frente a los archivos PCML serializados

Tome en consideración el siguiente código para construir un objeto ProgramCallDocument:

```
    AS400 as400 = new AS400();
    ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400,
"com.mycompany.mypackage.myPcmlDoc");
```

El constructor de ProgramCallDocument intentará en primer lugar encontrar un archivo PCML serializado cuyo nombre sea myPcmlDoc.pcml.ser en el paquete com.mycompany.mypackage de la CLASSPATH Java. Si no existe un archivo PCML serializado, el constructor intentará encontrar un archivo fuente PCML llamado myPcmlDoc.pcml en el paquete com.mycompany.mypackage de la CLASSPATH Java. Si tampoco existe ese archivo fuente PCML, se lanzará una excepción.

Nombres calificados

La aplicación Java utiliza ProgramCallDocument.setValue() para establecer los valores de entrada para el programa de iSeries al que se llama. Del mismo modo, la aplicación utiliza ProgramCallDocument.getValue() para recuperar los valores de salida del programa de iSeries.

Cuando acceda a los valores desde la clase ProgramCallDocument, debe especificar el nombre totalmente calificado del elemento documento o del código <**data**>.El nombre calificado es una concatenación de los nombres de todos los códigos continentes, en la que los nombres se separan mediante un punto.

Por ejemplo, si suponemos el siguiente fuente PCML, el nombre calificado del elemento "**nbrPolygons**" sería "**polytest.parm1.nbrPolygons**".El nombre calificado para acceder al valor "**x**" de uno de los puntos de uno de los polígonos sería "**polytest.parm1.polygon.point.x**".

Si alguno de los elementos necesarios para formar el nombre calificado no tiene nombre, los descendientes de ese elemento no tendrían un nombre calificado. Desde el programa Java no puede accederse a ningún elemento que no tenga un nombre calificado.

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- El parámetro 1 contiene una cuenta (nº total) de polígonos junto
con una matriz de polígonos -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Cada polígono contiene una cuenta del número de puntos junto con
una matriz de puntos -->
      <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>
```

Acceso a los datos de las matrices

Cualquier elemento <**data**> o <**struct**> se puede definir como matriz si se utiliza el atributo **count**. O bien, un elemento <**data**> o <**struct**> puede estar contenido dentro de otro elemento <**struct**> que esté definido como matriz.

Además, un elemento <**data**> o <**struct**> puede estar en una matriz multidimensional si más de un elemento continente tiene especificado un atributo **count**.

Para que su aplicación establezca u obtenga valores definidos como una matriz o definidos dentro de una matriz, debe especificar el índice de matriz para cada una de las dimensiones de la matriz. Los índices de matriz se pasan en forma de una matriz de valores **int**. Siguiendo con el fuente anterior de la matriz de polígonos, puede utilizarse el siguiente código Java para recuperar la información sobre los polígonos:

```
    ProgramCallDocument polytest; // Inicializado en algún otro lugar.
    Integer nbrPolygons, nbrPoints, pointX, pointY;
    nbrPolygons = (Integer) polytest.getValue("polytest.parm1.nbrPolygons");
    System.out.println("Número de polígonos:" + 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("  Número de puntos:" + 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);
        }
    }
```

Depuración

Al utilizar el lenguaje PCML para llamar a programas que tengan estructuras de datos complejas, es fácil que el PCML contenga errores que provoquen excepciones de la clase ProgramCallDocument. Si los errores están relacionados con una descripción incorrecta de los desplazamientos y longitudes de los datos, la depuración de las excepciones puede resultar complicada.

Utilice el método siguiente de la clase Trace para activar el rastreo PCML:

```
Trace.setTraceOn(true);      // Active la función de rastreo.
Trace.setTracePcmlOn(true);  // Active el rastreo PCML.
```

Nota: todos los métodos públicos de la clase PcmlMessageLog, incluido el de rastreo, han quedado obsoletos en la versión V5R2.

El método setFileName() de Trace permite enviar los siguientes tipos de información a archivos de anotaciones específicos o, por omisión, a System.out:

- Un vuelco de los datos hexadecimales que se estén transfiriendo entre la aplicación Java y el programa de iSeries. Este tipo de información muestra los parámetros de entrada del programa después de que los datos de tipo carácter se hayan convertido a EBCDIC y de que los enteros se hayan convertido a big-endian. También muestra los parámetros de salida antes de que se hayan convertido al entorno Java.

  Los datos aparecen en un formato típico de vuelco hexadecimal, con los dígitos hexadecimales a la izquierda y una interpretación de tipo carácter a la derecha. A continuación figura un ejemplo de este formato de vuelco:

  ```
  qgyolobj[6]
  Offset : 0....... 4....... 8....... C....... 0....... 4....... 8.......
  C.......   0...4...8...C...0...4...8...C...
       0 : 5CE4E2D9 D7D9C640 4040
  **USRPRF                         *
  ```

  En el ejemplo anterior, el vuelco indica que el séptimo parámetro tiene 10 bytes de datos establecidos en "*USRPRF ".

- En el caso de los parámetros de salida, a continuación del vuelco hexadecimal hay una descripción de cómo se han interpretado los datos para el documento.

  ```
  /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:
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

Los mensajes anteriores pueden resultar de gran utilidad para diagnosticar los casos en que los datos de salida procedentes del programa de iSeries no coinciden con el fuente PCML. Estos errores pueden producirse fácilmente cuando se utilizan longitudes y desplazamientos dinámicos.

Sintaxis de PCML

El lenguaje PCML consta de los códigos siguientes, cada uno de ellos con sus propios códigos de atributos:

- El código program señala el principio y el final del código que describe un programa.
- El código struct define una estructura con nombre que puede especificarse como argumento en un programa o como campo dentro de otra estructura con nombre. Un código de estructura contiene un código de datos o de estructura para cada uno de los campos de la estructura.
- El código data define un campo dentro de un programa o de una estructura.

En el ejemplo siguiente, la sintaxis de PCML describe un programa con una categoría de datos y algunos datos aislados.

```
<program>

    <struct>
    <data> </data>
      </struct>

    <data> </data>

  </program>
```

Código PCML program

El código PCML program puede ampliarse con los elementos siguientes:

```
<program name="nombre"
    [ entrypoint="nombre-punto-entrada" ]
»   [ epccsid="ccsid" ]«
    [ path="nombre-vía" ]
    [ parseorder="lista-nombres" ]
    [ returnvalue="{ void | integer }" ]
    [ threadsafe="{ true | false }" ]>
  </program>
```

En la siguiente tabla figuran los atributos del código program. Cada entrada contiene el nombre de atributo, los valores válidos posibles y una descripción del atributo.

Atributo	Valor	Descripción
entrypoint=	*nombre-punto-entrada*	Especifica el nombre del punto de entrada dentro de un objeto de programa de servicio que es el destino de esta llamada a programa.
»**epccsid=**	*ccsid*	Especifica el CCSID del punto de entrada dentro de un programa de servicio. Para obtener más información, consulte las notas de las entradas de programa de servicio en el javadoc de ServiceProgramCall. «
name=	*nombre*	Especifica el nombre del programa.
path=	*nombre-vía*	Especifica la vía de acceso al objeto programa. El valor por omisión es presuponer que el programa está en la biblioteca QSYS. La vía debe ser un nombre válido de vía IFS de acceso a un objeto *PGM o *SRVPGM. Si se llama a un objeto *SRVPGM, se debe especificar el atributo de punto de entrada (entrypoint) para indicar el nombre del punto de entrada al que se ha de llamar. Si no se especifica el atributo entrypoint, el valor por omisión de este atributo consiste en presuponer que se trata de un objeto *PGM de la biblioteca QSYS. Si se especifica el atributo entrypoint, el valor por omisión de este atributo consiste en presuponer que se trata de un objeto *SRVPGM de la biblioteca QSYS. El nombre de la vía se debe especificar con todos los caracteres en mayúsculas. »No utilice el atributo **path** cuando la aplicación tenga que establecer la vía de acceso en tiempo de ejecución, como por ejemplo cuando un usuario especifica qué biblioteca se utiliza para la instalación. En este caso, utilice el método ProgramCallDocument.setPath().«

parseorder=	*lista-nombres*	Especifica el orden en que se procesarán los parámetros de salida. El valor especificado es una lista de nombres de parámetros separados mediante espacios en blanco y escritos en el orden en que se han de procesar. Los nombres de la lista deben ser idénticos a los especificados en el atributo **name** de los códigos que pertenecen al código **<program>**. El valor por omisión consiste en procesar los parámetros de salida en el orden en que aparecen los códigos en el documento. Algunos programas devuelven información en un parámetro que describe información de un parámetro anterior. Por ejemplo, imagine que un programa devuelve una matriz de estructuras en el primer parámetro y el número de entradas de la matriz en el segundo parámetro. En este caso, es preciso procesar primero el segundo parámetro para que ProgramCallDocument pueda averiguar cuántas estructuras se han de procesar en el primer parámetro.
returnvalue=	*void* El programa no devuelve ningún valor. *integer* El programa devuelve un entero de 4 bytes con signo.	Especifica el tipo de valor que se devuelve (si es que se devuelve algún valor) desde una llamada a programa de servicio. Este atributo no está permitido para llamadas a objetos *PGM.
threadsafe=	*true* El programa se considera seguro en ejecución multihebra. *false* El programa no es seguro en ejecución multihebra.	Si llama a un programa Java y a un programa de iSeries que se encuentran en el mismo servidor, utilice esta propiedad para especificar si desea llamar al programa de iSeries en el mismo trabajo y en la misma hebra que utiliza el programa Java. Si sabe que el programa es seguro en ejecución multihebra, al establecer la propiedad en *true* obtendrá un mejor rendimiento. Para mantener la seguridad del entorno, por omisión se llama a los programas en trabajos servidores aparte. El valor por omisión es *false*.

Código PCML struct

El código PCML struct puede ampliarse con los elementos siguientes:

```
<struct name="nombre"
    [ count="{número | nombre-datos }"]
    [ maxvrm="serie-versión" ]
    [ minvrm="serie-versión" ]
    [ offset="{número | nombre-datos }" ]
    [ offsetfrom="{ número | nombre-datos | nombre-estructura }" ]
    [ outputsize="{número | nombre-datos }" ]
    [ usage="{ inherit | input | output | inputoutput }" ]>
</struct>
```

En la siguiente tabla figuran los atributos del código struct. Cada entrada contiene el nombre de atributo, los valores válidos posibles y una descripción del atributo.

Atributo	Valor	Descripción
name=	*nombre*	Especifica el nombre del elemento **<struct>.**
count=	*número* donde *número* define una matriz dimensionada fija. *nombre-datos* donde *nombre-datos* define el nombre de un elemento **<data>** dentro del documento PCML que contendrá, en tiempo de ejecución, el número de elementos de la matriz. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento **<data>** que se haya definido con la especificación **type=***"int"*. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica que el elemento es una matriz e identifica el número de entradas de la matriz. Si se omite este atributo, el elemento no está definido como una matriz, aunque puede encontrarse dentro de otro elemento que esté definido como matriz.
maxvrm=	*serie-versión*	Especifica la versión de OS/400 superior en la que existe el elemento. Si la versión de OS/400 es superior a la especificada en el atributo, el elemento y sus hijos, si es que existen, no se procesarán durante una llamada a un programa. El atributo **maxvrm** resulta de utilidad para definir interfaces de programa que difieren entre releases de OS/400. La sintaxis de la serie de versión debe ser "VvRrMm", donde las letras mayúsculas "V", "R" y "M" son caracteres literales y las minúsculas "v", "r" y "m" son uno o varios dígitos que representan la versión, el release y el nivel de modificación, respectivamente. El valor de "v" debe estar comprendido entre 1 y 255, ambos inclusive. El valor de "r" y "m" debe estar comprendido entre 0 y 255, ambos inclusive.

minvrm=	*serie-versión*	Especifica la versión de OS/400 inferior en la que existe este elemento. Si la versión de OS/400 es inferior a la especificada en este atributo, este elemento y sus hijos, si es que existen, no se procesarán durante una llamada a un programa. Este atributo resulta de utilidad para definir interfaces de programa que difieren entre releases de OS/400. La sintaxis de la serie de versión debe ser "VvRrMm", donde las letras mayúsculas "V", "R" y "M" son caracteres literales y las minúsculas "v", "r" y "m" son uno o varios dígitos que representan la versión, el release y el nivel de modificación, respectivamente. El valor de "v" debe estar comprendido entre 1 y 255, ambos inclusive. El valor de "r" y "m" debe estar comprendido entre 0 y 255, ambos inclusive.
offset=	*número* donde *número* define un desplazamiento fijo. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento PCML que contendrá, en tiempo de ejecución, el desplazamiento para el elemento. El **nombre-datos** especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con la especificación **type=***"int"*. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica el desplazamiento para el elemento <**struct**> dentro de un parámetro de salida. Algunos programas devuelven información con una estructura fija seguida de uno o más campos o estructuras de longitud variable. En este caso, la ubicación de un elemento de longitud variable se especifica normalmente como desplazamiento o desplazamiento entre estructuras dentro del parámetro. El atributo **offset** se utiliza para describir el desplazamiento para este elemento <**struct**>. El atributo **offset** se utiliza junto con el atributo **offsetfrom**. Si no se especifica el atributo **offsetfrom**, la ubicación base para el desplazamiento especificado en el atributo **offset** es el padre del elemento. Si desea obtener más información sobre cómo se utilizan los atributos **offset** y **offsetfrom**, consulte Especificación de desplazamientos. Los atributos **offset** y **offsetfrom** solo se utilizan para procesar los datos de salida de un programa. Estos atributos no controlan el desplazamiento ni el desplazamiento entre estructuras de los datos de entrada. Si se omite este atributo, la ubicación de los datos de este elemento sigue de inmediato al elemento anterior del parámetro, si existe.

offsetfrom=	*número* donde *número* define una ubicación base fija. El atributo *número* se utiliza normalmente para especificar **number=***"0"*, lo cual indica que se trata de un desplazamiento absoluto contado a partir del principio del parámetro. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> que se utilizará como ubicación base para el desplazamiento. El nombre de elemento especificado debe ser el padre o un ancestro de este elemento. El valor del atributo **offset** será relativo a la ubicación del elemento que se especifica en este atributo. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un ancestro de este elemento. Si desea obtener más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos. *nombre-estructura* donde *nombre-estructura* define el nombre de un elemento <**struct**> que se ha de utilizar como ubicación base del desplazamiento. El nombre de elemento especificado debe ser el padre o un ancestro de este elemento. El valor del atributo **offset** será relativo a la ubicación del elemento que se especifica en este atributo. El *nombre-estructura* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un ancestro de este elemento. Si desea obtener más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica la ubicación base que el atributo **offset** toma como referencia. Si no se especifica el atributo **offsetfrom**, la ubicación base del desplazamiento especificado en el atributo **offset** es el padre de este elemento. Si desea obtener más información sobre cómo se utilizan los atributos **offset** y **offsetfrom**, consulte Especificación de desplazamientos. Los atributos **offset** y **offsetfrom** solo se utilizan para procesar los datos de salida de un programa. Estos atributos no controlan el desplazamiento ni el desplazamiento entre estructuras de los datos de entrada.
outputsize=	*número* donde *número* define un número fijo de bytes que se han de reservar. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento PCML que contendrá, en tiempo de ejecución, el número de bytes que se han de reservar para los datos de salida. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con la especificación **type=***"int"*. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica el número de bytes que se han de reservar para los datos de salida del elemento. En el caso de los parámetros de salida de longitud variable, se necesita el atributo **outputsize** para especificar cuántos bytes deben reservarse para los datos que se han de devolver desde el programa del servidor. Puede especificarse un atributo **outputsize** en todos los campos de longitud variable y en todas las matrices de tamaño variable, o bien puede especificarse para un parámetro completo que contenga uno o más campos de longitud variable. El atributo **outputsize** no es necesario y no debería especificarse en el caso de los parámetros de salida de tamaño fijo. El valor especificado en este atributo se utiliza como tamaño total del elemento, incluidos todos sus hijos. Por consiguiente, el atributo **outputsize** no se tiene en cuenta en los hijos ni

		en los descendientes del elemento. Si se omite este atributo, el número de bytes que se han de reservar para los datos de salida se determina en tiempo de ejecución sumando el número de bytes que se han de reservar para todos los hijos del elemento <**struct**>.
usage=	*inherit*	La utilización se hereda del elemento padre. Si la estructura no tiene padre, se supone que la utilización es **inputoutput**.
	input	La estructura es un valor de entrada dirigido al programa del sistema principal. Si los datos son de tipo carácter o numérico, se efectúa la conversión adecuada.
	output	La estructura es un valor de salida procedente del programa del sistema principal. Si los datos son de tipo carácter o numérico, se efectúa la conversión adecuada.
	inputoutput	La estructura es tanto un valor de entrada como uno de salida.

Especificación de desplazamientos

Algunos programas devuelven información con una estructura fija seguida de uno o más campos o estructuras de longitud variable. En este caso, la ubicación de un elemento de longitud variable se especifica normalmente como desplazamiento o desplazamiento entre estructuras dentro del parámetro.

Un desplazamiento es la distancia en bytes desde el principio de un parámetro hasta el principio de un campo o de una estructura. Un desplazamiento entre estructuras es la distancia en bytes desde el principio de una estructura hasta el principio de otra estructura.

En el caso de los primeros, dado que la distancia se cuenta desde el principio del parámetro, se debería especificar **offsetfrom="0"**. A continuación se muestra un ejemplo de desplazamiento desde el principio del parámetro:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variable receiver contiene una vía de acceso -->
    <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>
```

En el caso de los desplazamientos entre estructuras, puesto que la distancia se cuenta desde el principio de otra estructura, se especifica el nombre de la estructura que es el punto de partida del desplazamiento. A continuación se muestra un ejemplo de un desplazamiento entre estructuras desde el principio de una estructura determinada:

```
<pcml ="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variable receiver contiene un objeto -->
    <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>
```

Código PCML data

El código PCML data puede tener los atributos que se indican más abajo. Los atributos delimitados por corchetes, [], son opcionales. Si especifica un atributo opcional, no incluya los corchetes en el fuente. A continuación se facilita una lista con algunos valores de atributos delimitados por llaves, {}, y las opciones posibles separadas por barras verticales, |. Cuando especifique uno de estos atributos, no incluya las llaves en el fuente y especifique únicamente una de las opciones que se muestran.

```
<data type="{ char | int | packed | zoned | float | byte | struct }"
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 |
DEFAULT }"]
    [ ccsid="{ número | nombre-datos }" ]
    [ chartype="{ onebyte | twobyte }"]
    [ count="{ número | nombre-datos }" ]
    [ init="serie" ]
    [ length="{ número | nombre-datos }" ]
    [ maxvrm="serie-versión" ]
    [ minvrm="serie-versión" ]
    [ name="nombre" ]
    [ offset="{ número | nombre-datos }" ]
    [ offsetfrom="{ número | nombre-datos | nombre-estructura }" ]
    [ outputsize="{ número | nombre-datos | nombre-estructura }" ]
    [ passby= "{ reference | value }" ]
    [ precision="número" ]
    [ struct="nombre-estructura" ]
    [ trim="{ right | left | both | none }" ]
    [ usage="{ inherit | input | output | inputoutput }" ]>
</data>
```

En la siguiente tabla figuran los atributos del código data. Cada entrada contiene el nombre de atributo, los valores válidos posibles y una descripción del atributo.

Atributo	Valor	Descripción
type=	*char* donde *char* indica un valor de tipo carácter. Un valor de datos de tipo *char* se devuelve como *java.lang.String*.Para obtener más información, consulte los valores *char* para la longitud. *int* donde *int* es un valor entero. Un valor de datos *int* se devuelve como *java.lang.Long*. Para obtener más información, consulte los valores *int* para la longitud y la precisión. *packed* donde *packed* es un valor decimal empaquetado. Un valor de datos *packed* se devuelve como *java.math.BigDecimal*. Para obtener más información, consulte los valores *packed* para la longitud y la precisión. *zoned* donde *zoned* es un valor decimal con zona. Un valor de datos *zoned* se devuelve como *java.math.BigDecimal*. Para obtener más información, consulte los valores *zoned* para la	Indica el tipo de datos que se utiliza (carácter, entero, empaquetado, con zona, coma flotante o estructura). Los valores de los atributos de longitud y precisión varían según los diferentes tipos de datos. Para obtener más información, consulte los valores de longitud y precisión.

	longitud y la precisión. *float* donde *float* es un valor de coma flotante. El atributo **length** especifica el número de bytes, "4" u "8". Un entero de 4 bytes se devuelve como *java.lang.Float*. Un entero de 8 bytes se devuelve como *java.lang.Double*. Para obtener más información, consulte los valores *float* para la longitud. *byte* donde *byte* es un valor de tipo byte. No se efectúa ninguna conversión de los datos. Un valor de datos *byte* se devuelve como una matriz de valores *byte* (*byte[]*). Para obtener más información, consulte los valores *byte* para la longitud. *struct* donde *struct* especifica el nombre del elemento <**struct**>. Un valor *struct* permite definir una estructura una vez y volver a utilizarla varias veces dentro del mismo documento. Cuando **type=***"struct"*, es como si la estructura especificada apareciera en ese lugar del documento. Un valor *struct* no permite un valor de longitud y no tiene ningún valor para la precisión.	
bidistringtype=	*DEFAULT* donde DEFAULT es el tipo serie por omisión para datos no bidireccionales (LTR). *ST4* donde *ST4* es el tipo de serie 4. *ST5* donde *ST5* es el tipo de serie 5. *ST6* donde *ST6* es el tipo de serie 6. *ST7* donde *ST7* es el tipo de serie 7. *ST8* donde *ST8* es el tipo de serie 8. *ST9* donde *ST9* es el tipo de serie 9. *ST10* donde *ST10* es el tipo de serie 10. *ST11* donde *ST11* es el tipo de serie 11.	Especifica el tipo de serie bidireccional para los elementos <**data**> que tengan **type=***"char"*. Si se omite este atributo, el tipo de serie para este elemento viene determinado por el CCSID (especificado explícitamente o el CCSID por omisión del entorno de sistema principal). Los tipos de serie están definidos en el javadoc correspondiente a la clase BidiStringType.

ccsid=	*número* donde *número* define un CCSID fijo, que no cambia nunca. *nombre-datos* donde *nombre-datos* define el nombre que contendrá, en tiempo de ejecución, el CCSID de los datos de tipo carácter. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con la especificación **type**=*"int"*. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica el CCSID (ID de juego de caracteres codificados) de los datos de tipo carácter para el elemento <**data**>. El atributo **ccsid** sólo se puede especificar para los elementos <**data**> que tengan **type**=*"char"*. Si se omite este atributo, se presupone que los datos de tipo carácter de este elemento tienen el CCSID por omisión del entorno de sistema principal.
chartype=	*onebyte* donde *onebyte* especifica el tamaño de cada carácter. *twobyte* donde *twobyte* especifica el tamaño de cada carácter. Cuando se utiliza *chartype*, el atributo **length**=*"número"* especifica el número de caracteres, no el número de bytes.	Especifica el tamaño de cada carácter.
count=	*número* donde *número* define un número fijo de elementos de una matriz dimensionada. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento PCML que contendrá, en tiempo de ejecución, el número de elementos de la matriz. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con la especificación **type**=*"int"*. Consulte Resolución de nombres relativos para obtener más información sobre cómo se resuelven los nombres relativos.	Especifica que el elemento es una matriz e identifica el número de entradas de la matriz. Si se omite el atributo *count*, el elemento no está definido como matriz, aunque puede encontrarse dentro de otro elemento que esté definido como matriz.
init=	*serie*	Especifica un valor inicial para el elemento <**data**>. Se utiliza el valor *init* si el programa de aplicación no establece de forma explícita ningún valor inicial al utilizar elementos <**data**> con la especificación **usage**=*"input"* o **usage**=*"inputoutput"*. El valor inicial especificado se utiliza para inicializar los valores escalares. Si el elemento está definido como matriz o se encuentra dentro de una estructura definida como matriz, el valor inicial especificado se utiliza como valor inicial para todas las entradas de la matriz.

length=	»*número* donde *número* define el número de bytes que requieren los datos. Sin embargo, al utilizar el atributo *chartype*, *número* especifica el número de caracteres, no el número de bytes.« *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento PCML que contendrá, en tiempo de ejecución, la longitud. Únicamente puede especificarse un *nombre-datos* para los elementos <**data**> que tengan **type**=*"char"* o **type**=*"byte"*. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con la especificación **type**=*"int"*. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica la longitud del elemento de datos. El uso de este atributo varía en función del tipo de datos. Para obtener más información, consulte los valores de longitud y precisión.
maxvrm=	*serie-versión*	Especifica la versión de iSeries superior en la que existe este elemento. Si la versión de iSeries es mayor que la especificada en este atributo, este elemento y sus hijos, en caso de que existan, no se procesarán durante una llamada a un programa. Este atributo resulta de utilidad para definir interfaces de programa que difieren entre releases de iSeries. La sintaxis de la serie de versión debe ser "VvRrMm", donde las letras mayúsculas "V", "R" y "M" son caracteres literales y las minúsculas "v", "r" y "m" son uno o varios dígitos que representan la versión, el release y el nivel de modificación, respectivamente. El valor de "v" debe estar comprendido entre 1 y 255, ambos inclusive. El valor de "r" y "m" debe estar comprendido entre 0 y 255, ambos inclusive.
minvrm=	*serie-versión*	Especifica la versión de iSeries inferior en la que existe este elemento. Si la versión de iSeries es menor que la especificada en este atributo, este elemento y sus hijos, en caso de que existan, no se procesarán durante una llamada a un programa. Este atributo resulta de utilidad para definir interfaces de programa que difieren entre releases de iSeries. La sintaxis de la serie de versión debe ser "VvRrMm", donde las letras mayúsculas "V", "R" y "M" son caracteres literales y las minúsculas "v", "r" y "m" son uno o varios dígitos que representan la versión, el release y el nivel de modificación, respectivamente. El valor de "v" debe estar comprendido entre 1 y 255, ambos inclusive. El valor de "r" y "m" debe estar comprendido entre 0 y 255, ambos inclusive.

name=	*nombre*	Especifica el nombre del elemento <**data**>.
offset=	*número* donde *número* define un desplazamiento fijo. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento PCML que contendrá, en tiempo de ejecución, el desplazamiento para este elemento. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con la especificación **type=***"int"*. Consulte Resolución de nombres relativos para obtener más información sobre cómo se resuelven los nombres relativos.	Especifica el desplazamiento para el elemento <**data**> dentro de un parámetro de salida. Algunos programas devuelven información con una estructura fija seguida de uno o más campos o estructuras de longitud variable. En este caso, la ubicación de un elemento de longitud variable se especifica normalmente como desplazamiento o desplazamiento entre estructuras dentro del parámetro. El atributo **offset** se utiliza junto con el atributo **offsetfrom**. Si no se especifica el atributo **offsetfrom**, la ubicación base del desplazamiento especificado en el atributo **offset** es el padre de este elemento. Si desea obtener más información sobre cómo se utilizan los atributos **offset** y **offsetfrom**, consulte Especificación de desplazamientos. Los atributos **offset** y **offsetfrom** solo se utilizan para procesar los datos de salida de un programa. Estos atributos no controlan el desplazamiento ni el desplazamiento entre estructuras de los datos de entrada. Si se omite este atributo, la ubicación de los datos de este elemento sigue de inmediato al elemento anterior en el parámetro, si existe.
offsetfrom=	*número* donde *número* define una ubicación base fija. *Número* se utiliza normalmente para especificar **number=***"0"*, lo cual indica que se trata de un desplazamiento absoluto contado a partir del principio del parámetro. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> que se utilizará como ubicación base para el desplazamiento. El nombre de elemento especificado debe ser el padre o un ancestro de este elemento. El valor del atributo **offset** será relativo a la ubicación del elemento que se especifica en este atributo. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un ancestro de este elemento. Si desea obtener más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos. *nombre-estructura* donde *nombre-estructura* define el nombre de un elemento <**struct**> que se utilizará como ubicación base para el desplazamiento. El nombre de elemento especificado debe ser el padre o un ancestro de este elemento. El valor del atributo	Especifica la ubicación base que el atributo **offset** toma como referencia. Si no se especifica el atributo **offsetfrom**, la ubicación base del desplazamiento especificado en el atributo **offset** es el padre de este elemento. Si desea obtener más información sobre cómo se utilizan los atributos **offset** y **offsetfrom**, consulte Especificación de desplazamientos. Los atributos **offset** y **offsetfrom** solo se utilizan para procesar los datos de salida de un programa. Estos atributos no controlan el desplazamiento ni el desplazamiento entre estructuras de los datos de entrada.

	offset será relativo a la ubicación del elemento que se especifica en este atributo. El *nombre-estructura* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un ancestro de este elemento. Si desea obtener más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	
outputsize=	*número* donde *número* define un número fijo de bytes que se han de reservar. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento PCML que contendrá, en tiempo de ejecución, el número de bytes que se han de reservar para los datos de salida. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con la especificación **type=***"int"*. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica el número de bytes que se han de reservar para los datos de salida del elemento. En el caso de los parámetros de salida de longitud variable, se necesita el atributo **outputsize** para especificar cuántos bytes deben reservarse para los datos que se han de devolver desde el programa de iSeries. Puede especificarse un atributo **outputsize** en todos los campos de longitud variable y en todas las matrices de dimensión variable, o bien puede especificarse para un parámetro completo que contenga uno o más campos de longitud variable. El atributo **outputsize** no es necesario y no debería especificarse en el caso de los parámetros de salida de tamaño fijo. El valor especificado en este atributo se utiliza como tamaño total del elemento, incluidos todos los hijos del elemento. Por consiguiente, el atributo **outputsize** no se tiene en cuenta en los hijos ni en los descendientes del elemento. Si se omite **outputsize**, el número de bytes que se han de reservar para los datos de salida se determina en tiempo de ejecución sumando el número de bytes que se han de reservar para todos los hijos del elemento <**struct**>.
passby=	*reference* donde *reference* indica que el parámetro se pasará por referencia. Cuando se llame al programa, se le pasará un puntero que señale al valor del parámetro. *value* donde *value* indica un valor de tipo entero. Este valor solo está permitido cuando se especifica **type=** *"int"* y **length=***"4"*.	Especifica si el parámetro se pasa por referencia o por valor. Este atributo sólo está permitido cuando este elemento es hijo de un elemento <**program**> que define una llamada a un programa de servicio.
precision=	*número*	Especifica el número de bytes de precisión para algunos tipos de datos numéricos. Para obtener más información, consulte los valores de longitud y precisión.
struct=	*nombre*	Especifica el nombre de un elemento <**struct**> para el elemento <**data**>. Un atributo **struct** solo se puede especificar para elementos <**data**> que tengan **type=***"struct"*.

trim=	*right* donde *right* es el valor por omisión que significa que se recortarán los espacios en blanco de cola. *left* donde *left* significa que se recortarán los espacios en blanco precedentes. *both* donde *both* significa que se recortarán los espacios en blanco precedentes y de cola. *none* donde *none* significa que no se recortarán los espacios en blanco.	Especifica cómo se recortará el espacio en blanco desde los datos de tipo carácter.
usage=	*inherit*	La utilización se hereda del elemento padre. Si la estructura no tiene padre, se supone que la utilización es *inputoutput*.
	input	Define un valor de entrada para el programa del sistema principal. Si los datos son de tipo carácter o numérico, se efectúa la conversión adecuada.
	output	Define un valor de salida desde el programa del sistema principal. Si los datos son de tipo carácter o numérico, se efectúa la conversión adecuada.
	inputoutput	Define tanto un valor de entrada como uno de salida.

Especificación de desplazamientos

Algunos programas devuelven información con una estructura fija seguida de uno o más campos o estructuras de longitud variable. En este caso, la ubicación de un elemento de longitud variable se especifica normalmente como desplazamiento o desplazamiento entre estructuras dentro del parámetro.

Un desplazamiento es la distancia en bytes desde el principio de un parámetro hasta el principio de un campo o de una estructura. Un desplazamiento entre estructuras es la distancia en bytes desde el principio de una estructura hasta el principio de otra estructura.

En el caso de los primeros, dado que la distancia se cuenta desde el principio del parámetro, se debería especificar **offsetfrom="0"**. A continuación se muestra un ejemplo de desplazamiento desde el principio del parámetro:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variable receiver contiene una vía de acceso -->
    <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>
```

En el caso de los desplazamientos entre estructuras, puesto que la distancia se cuenta desde el principio de otra estructura, se especifica el nombre de la estructura que es el punto de partida del desplazamiento. A continuación se

muestra un ejemplo de un desplazamiento entre estructuras desde el principio de una estructura determinada:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- la variable receiver contiene un objeto -->
    <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>
```

Valores de longitud y precisión

Los valores de los atributos de longitud y precisión varían según los diferentes tipos de datos. En la tabla siguiente se indica cada tipo de datos con una descripción de los valores posibles de longitud y precisión.

Tipo de datos	Longitud	Precisión
type="char"	Número de bytes de datos de este elemento, que puede ser distinto del número de caracteres. Debe especificar un *número* literal o un *nombre de datos*.	No válido.
type="int"	Número de bytes de datos de este elemento: 2, 4 o 8. Debe especificar un *número* literal.	Indica el número de bits de precisión y si el entero es con signo o sin signo: ● Para **length**="2" ❍ Utilice **precision**="15" para un entero de 2 bytes con signo. Este es el valor por omisión. ❍ Utilice **precision**="16" para un entero de 2 bytes sin signo. ● Para **length**="4" ❍ Utilice **precision**="31" para un entero de 4 bytes con signo. ❍ Utilice **precision**="32" para un entero de 4 bytes sin signo. ● Para **length**="8" utilice **precision**="63" para un entero de 8 bytes con signo.
type="packed" o "zoned"	Número de dígitos numéricos de datos de este elemento. Debe especificar un *número* literal.	Número de dígitos decimales del elemento. Este número debe ser superior o igual a cero e inferior o igual al número total de dígitos especificado en el atributo **length**.
type="float"	Número de bytes, 4 o 8, de datos de este elemento. Debe especificar un *número* literal.	No válido.
type="byte"	Número de bytes de datos de este elemento. Debe especificar un *número* literal o un *nombre de datos*.	No válido.
type="struct"	No permitido.	No válido.

Resolución de nombres relativos

Hay varios atributos que permiten especificar como valor de atributo el nombre de otro elemento, o código, dentro del documento. El nombre especificado puede ser un nombre totalmente calificado o un nombre relativo al código actual.

Para resolver un nombre se mira si el nombre corresponde al nombre de un hijo o un descendiente del código que

contiene el código actual. Si el nombre no puede resolverse a este nivel, la búsqueda continúa en el código continente que le sigue en la jerarquía. Este proceso de resolución continúa hasta que se llega, en última instancia, a una coincidencia con un código que esté contenido en el código **<pcml>** o el código <**rfml**>, en cuyo caso el nombre se considera absoluto, no relativo.

A continuación se muestra un ejemplo de PCML:

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- El parámetro 1 contiene una cuenta (nº total) de polígonos junto
con una matriz de polígonos -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Cada polígono contiene una cuenta del número de puntos junto con
una matriz de puntos -->
      <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>
```

A continuación se muestra un ejemplo de RFML:

```
<rfml version="4.0">
  <struct name="polygon">
    <!-- Cada polígono contiene una cuenta del número de puntos junto con
una matriz de puntos -->
    <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">
    <!-- Este formato contiene una cuenta de polígonos junto con una matriz
de polígonos -->
    <data name="nbrPolygons" type="int" length="4" init="5" />
    <data name="polygon" type="struct" struct="polygon" count="nbrPolygons"
/>
  </recordformat>
</rfml>
```

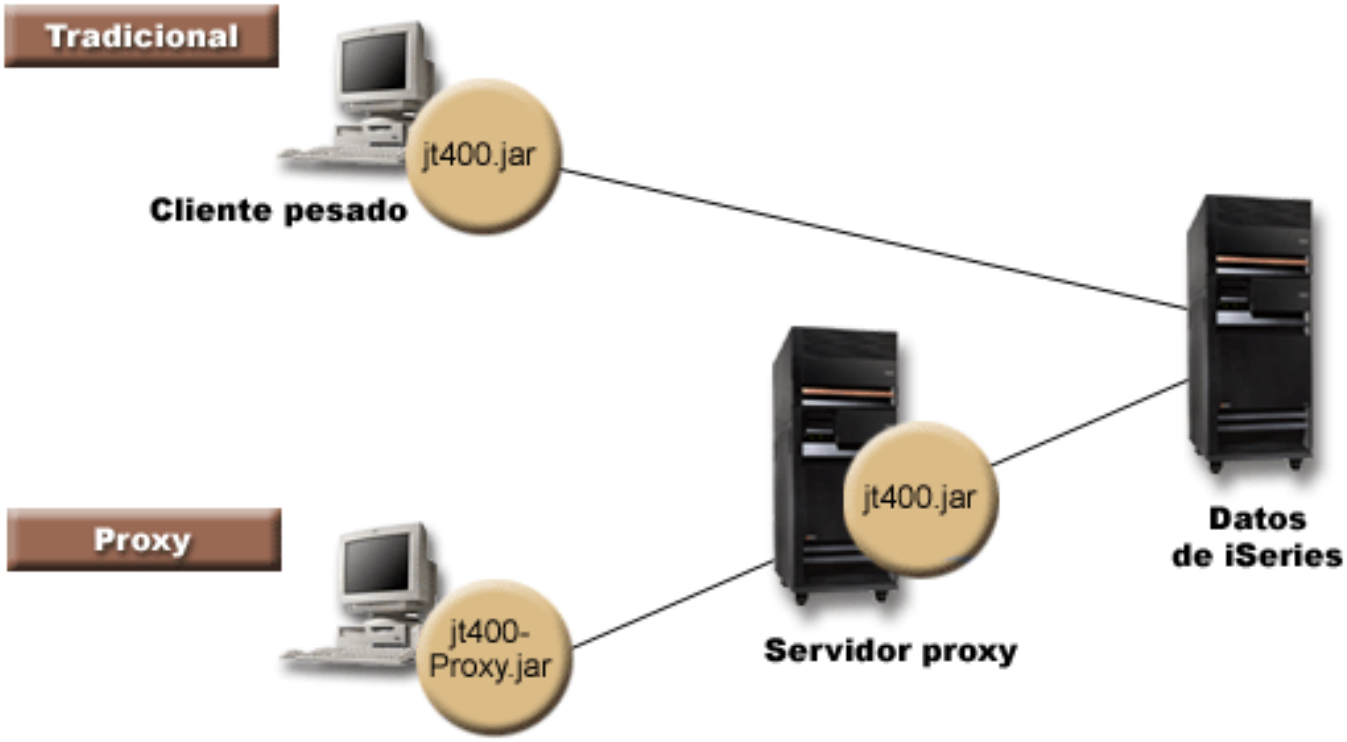
Soporte de proxy

IBM Toolbox para Java incluye soporte de proxy para algunas clases. El soporte de proxy es el proceso que IBM Toolbox para Java necesita realizar para llevar a cabo una tarea en una máquina virtual Java (JVM) cuando la aplicación se encuentra en una máquina virtual Java distinta. El soporte de proxy incluye utilizar el protocolo SSL (capa de sockets segura) para cifrar datos.

Las clases de proxy residen en jt400Proxy.jar, que se suministra con el resto de IBM Toolbox para Java. Las clases de proxy, como las demás clases de IBM Toolbox para Java, están formadas por un conjunto de clases Java independientes de la plataforma que pueden ejecutarse en cualquier sistema con una máquina virtual Java. Las clases de proxy despachan todas las llamadas a método a una aplicación de servidor o a un servidor proxy. Las clases del producto IBM Toolbox para Java completo están en el servidor proxy. Cuando un cliente utiliza una clase de proxy, la petición se transfiere al servidor proxy que crea y administra los objetos reales de IBM Toolbox para Java.

La figura 1 muestra cómo se conectan al servidor el cliente estándar y el cliente proxy. El servidor proxy puede ser el iSeries que contiene los datos.

Figura 1: cómo se conectan a un servidor un cliente estándar y un cliente proxy



Una aplicación que utiliza el soporte de proxy se ejecuta con mayor lentitud que al utilizar las clases de IBM Toolbox para Java estándar debido a la comunicación adicional necesaria para dar soporte a las clases de proxy más pequeñas. Las aplicaciones que realizan menos llamadas a método sufren un menor deterioro del rendimiento.

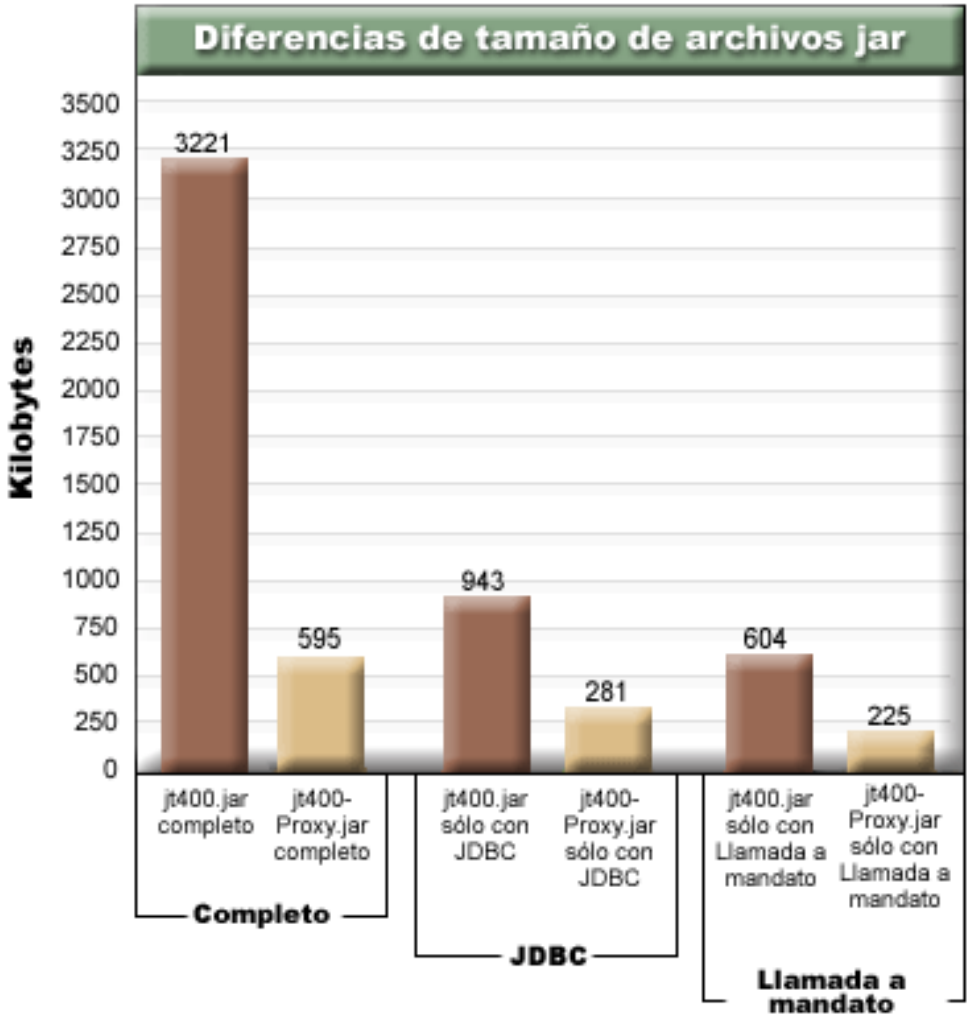
Antes del soporte de proxy, las clases que contenían la interfaz pública, todas las clases necesarias para procesar una petición y la propia aplicación se ejecutaban en la misma máquina virtual Java. Al utilizar el soporte de proxy, la interfaz pública debe estar con la aplicación, pero las clases para procesar las peticiones pueden ejecutarse en una máquina virtual Java distinta. El soporte de proxy no cambia la interfaz pública. Un mismo programa puede ejecutarse con la versión de proxy de IBM Toolbox para Java o con la versión estándar.

Cómo utilizar el archivo jt400Proxy.jar

El objetivo del escenario proxy multinivel es conseguir que el archivo jar de interfaz pública sea lo más pequeño posible para que el proceso de bajar este archivo de un applet dure menos tiempo. Cuando se utilizan las clases de proxy, no es necesario instalar en el cliente todo el producto IBM Toolbox para Java. En su lugar, utilice AS400JarMaker en el archivo jt400Proxy.jar para incluir únicamente los componentes necesarios, lo que hace que el archivo jar sea lo más pequeño posible.

La figura 2 compara el tamaño de los archivos jar de proxy con el de los archivos jar estándar:

Figura 2: comparación del tamaño de los archivos jar de proxy y los archivos jar estándar



Una ventaja adicional es que el soporte de proxy requiere tener menos puertos abiertos a través de un cortafuegos. Con IBM Toolbox para Java estándar, debe tener varios puertos abiertos. Ello se debe a que cada servicio de IBM Toolbox para Java emplea un puerto distinto para comunicarse con el servidor. Por ejemplo, la llamada a mandato emplea un puerto distinto del que emplea JDBC, que a su vez utiliza un puerto distinto del que emplea la impresión, etc. Cada uno de estos puertos debe estar permitido a través del cortafuegos. Sin embargo, al utilizar el soporte de proxy, todos los datos fluyen por el mismo puerto.

Proxy estándar y túneles HTTP

Hay disponibles dos opciones para llevar a cabo la ejecución mediante un proxy, que son proxy estándar y túneles HTTP:

- La comunicación de proxy estándar es aquella en que el cliente proxy y el servidor proxy se comunican utilizando un socket en un puerto. El puerto por omisión es 3470. Para cambiar el puerto por omisión, puede

elegir entre utilizar el método setPort() en la clase ProxyServer o emplear la opción -port al iniciar el servidor proxy. Por ejemplo:

```
java com.ibm.as400.access.ProxyServer -port 1234
```

- La comunicación de túneles HTTP es aquella en que el cliente proxy y el servidor proxy se comunican mediante el servidor HTTP. IBM Toolbox para Java proporciona un servlet que maneja la petición de proxy. El cliente proxy llama al servlet mediante el servidor HTTP. La ventaja de utilizar los túneles es que no es necesario abrir un puerto adicional a través de los cortafuegos, ya que la comunicación se efectúa mediante el puerto HTTP. La desventaja de la comunicación por túneles es que es más lenta que la opción de proxy estándar.

IBM Toolbox para Java utiliza el nombre de servidor proxy para determinar si se utiliza la opción de proxy estándar o proxy por túneles:

- En el caso de proxy estándar, utilice únicamente el nombre del servidor. Por ejemplo:

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

- En el caso de proxy por túneles, utilice un URL para forzar al cliente proxy a utilizar los túneles. Por ejemplo:

```
com.ibm.as400.access.AS400.proxyServer=http://myServer
```

Cuando se ejecuta proxy estándar, existe una conexión por socket entre el cliente y el servidor. Si esa conexión falla, el servidor elimina los recursos asociados a ese cliente.

Cuando se utilizan los túneles HTTP, al utilizar el protocolo HTTP el proxy no tiene conexión. Esto significa que se establece una conexión nueva para cada flujo de datos. Dado que el protocolo es sin conexión, el servidor no sabe si la aplicación de cliente ya no está activa. Por consiguiente, el servidor no sabe cuándo eliminar los recursos. El servidor de comunicación por túneles resuelve este problema usando una hebra para eliminar los recursos en un intervalo predeterminado (que se basa en un valor de tiempo de espera).

Una vez transcurrido el intervalo predeterminado, se ejecuta la hebra que elimina los recursos que no se han utilizado últimamente. Dos propiedades del sistema rigen la hebra:

- com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval indica con qué frecuencia, en segundos, se ejecuta la hebra de limpieza. El valor por omisión es cada dos horas.
- com.ibm.as400.access.TunnelProxyServer.clientLifetime indica cuánto tiempo, en segundos, puede estar desocupado un recurso antes de que se elimine. El valor por omisión es 30 minutos.

Cómo se utiliza el servidor proxy

Para utilizar la implementación de servidor proxy de las clases de IBM Toolbox para Java, siga estos pasos:

1. Ejecute AS400ToolboxJarMaker en jt400Proxy.jar para descartar las clases que no necesite. Este paso es opcional pero recomendado.
2. Determine cómo ha de poner el archivo jt400Proxy.jar en el cliente.
   - En el caso de los programas Java, emplee la clase AS400ToolboxInstaller u otro método para poner el archivo en el cliente.
   - En el caso de los applets Java, puede bajar el archivo jar del servidor HTML.
3. Determine qué servidor va a utilizar para que haga de servidor proxy.
   - En el caso de las aplicaciones Java, el servidor proxy puede ser cualquier equipo.
   - En el caso de los applets Java, el servidor proxy debe ejecutarse en el mismo equipo que el servidor HTTP.
4. Compruebe que haya colocado jt400.jar en la variable CLASSPATH del servidor.
5. Inicie el servidor proxy o utilice el servlet de proxy:

- Para proxy estándar, inicie el servidor proxy con el mandato siguiente:

```
java com.ibm.as400.access.ProxyServer
```

- Para proxy por túneles, configure el servidor HTTP para utilizar el servlet de proxy. El nombre de clase de servlet es com.ibm.as400.access.TunnelProxyServer y se incluye en jt400.jar.

6. En el cliente, establezca una propiedad del sistema para identificar el servidor proxy. IBM Toolbox para Java utiliza esta propiedad del sistema para determinar si se utiliza la opción de proxy estándar o proxy por túneles.
   - Para proxy estándar, el valor de la propiedad es el nombre de la máquina que ejecuta el servidor proxy. Por ejemplo:

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

   - En el caso de proxy por túneles, utilice un URL para forzar al cliente proxy a utilizar los túneles. Por ejemplo:

```
com.ibm.as400.access.AS400.proxyServer=http://myServer
```

7. Ejecute el programa cliente.

Si desea trabajar tanto con las clases de proxy como con las clases que no están en el archivo jt400Proxy.jar, puede hacer referencia al archivo jt400.jar, en vez de al archivo jt400Proxy.jar. El archivo jt400Proxy.jar es un subconjunto del archivo jt400.jar y por eso todas las clases de proxy están contenidas en el archivo jt400.jar.

Cómo se utiliza SSL

Al utilizar proxy, hay tres opciones disponibles para cifrar los datos a medida que fluyen del cliente proxy al servidor iSeries destino. Se utilizan varios algoritmos de SSL para cifrar datos.

1. Los flujos de datos entre el cliente proxy y el servidor proxy pueden cifrarse.
2. Los flujos de datos entre el servidor proxy y el servidor iSeries destino pueden cifrarse.
3. Los casos 1 y 2. El flujo de datos entre el cliente proxy y el servidor proxy y el flujo entre el servidor proxy y el iSeries destino pueden cifrarse.

Consulte SSL (capa de sockets segura) para obtener más información.

Ejemplos: cómo se utilizan los servidores proxy

Hemos proporcionado tres ejemplos específicos de cómo se utiliza un servidor proxy siguiendo los pasos que figuran más arriba.

- Ejecutar una aplicación Java utilizando el soporte de proxy
- Ejecutar un applet Java utilizando el soporte de proxy
- Ejecutar una aplicación Java utilizando el soporte de proxy por túneles

Clases habilitadas para trabajar con el servidor proxy

Algunas de las clases de IBM Toolbox para Java están habilitadas para trabajar con la aplicación de servidor proxy. Son las siguientes:

- JDBC
- Acceso a nivel de registro
- Sistema de archivos integrado
- Imprimir

- Colas de datos
- Llamada a mandato
- Llamada a programa
- Llamada a programa de servicio
- Espacio de usuario
- Área de datos
- Clase AS400
- Clase SecureAS400

Por ahora, las demás clases no están soportadas por jt400Proxy. Además, los permisos del sistema de archivos integrado no funcionan únicamente con el archivo jar de proxy. Sin embargo, puede utilizar la clase JarMaker para incluir estas clases a partir del archivo jt400.jar.

»Record Format Markup Language

RFML (Record Format Markup Language) es una extensión de XML para especificar formatos de registro. El componente RFML de IBM Toolbox para Java permite a las aplicaciones Java utilizar documentos RFML para especificar y manipular campos dentro de determinados tipos de registros.

Los documentos RFML, llamados archivos fuente RFML, representan un subconjunto de los tipos de datos de especificación de descripción de datos (DDS) de utilidad definidos para los archivos físicos y lógicos en los sistemas iSeries. Puede utilizar documentos RFML para gestionar la información de los elementos siguientes:

- Registros de archivo
- Entradas de cola de datos
- Espacios de usuario
- Almacenamientos intermedios de datos arbitrarios

Nota: para obtener más información sobre cómo se utiliza DDS para describir atributos de datos, consulte la referencia de DDS.

RFML guarda un enorme parecido con PCML (Program Call Markup Language), otra extensión de XML soportada por Toolbox para Java. RFML no es un subconjunto ni un superconjunto de PCML, sino un tipo de lenguaje de iguales que añade varios elementos y atributos nuevos y omite otros.

PCML proporciona una alternativa orientada a XML al uso de las clases ProgramCall y ProgramParameter. Del mismo modo, RFML ofrece una alternativa de fácil manejo y mantenimiento a las clases Record, RecordFormat y FieldDescription.

Encontrará más información sobre RFML en estos temas:

Requisitos
Averigüe cuáles son los requisitos de uso de RFML.

Ejemplo de RFML
Consulte cómo el uso de RFML en una aplicación reduce la cantidad (y en ocasiones la complejidad) del código que tiene que escribir. El ejemplo contiene un archivo fuente RFML de ejemplo.

Clase RecordFormatDocument
Consulte cómo utilizar la clase RecordFormatDocument con otras clases de Toolbox para Java para leer y escribir datos.

Documentos RFML y sintaxis RFML
Obtenga información sobre los documentos RFML, denominados archivos fuente RFML, y la sintaxis RFML tal como se ha definido en la definición de tipo de datos RFML.

RFML sólo es una forma de utilizar XML con el servidor. Para obtener más información sobre cómo utilizar XML con los servidores iSeries, consulte las extensiones XML de Toolbox para Java y XML (Extensible Markup Language).«

»Requisitos para utilizar RFML

El componente RFML tiene los mismos requisitos de máquina virtual Java de la estación de trabajo que el resto de IBM Toolbox para Java.

Además, para analizar RFML en tiempo de ejecución, la CLASSPATH de la aplicación debe incluir un analizador XML. El analizador XML debe ampliar la clase org.apache.xerces.parsers.SAXParser. Toolbox para Java contiene un analizador compatible, el analizador XML para Java, que se incluye en el archivo x4j400.jar de Toolbox para Java. Encontrará más información en Archivos jar.

Nota: RFML tiene los mismos requisitos de analizador que PCML. Como sucede con PCML, si preserializa el archivo RFML, no es necesario que incluya un analizador XML en la CLASSPATH de la aplicación para ejecutar la aplicación.«

»Ejemplo: cómo se utiliza RFML en comparación con el uso de las clases Record de Toolbox para Java

Este ejemplo muestra las diferencias entre el uso de RFML y el uso de las clases Record de Toolbox para Java.

Al utilizar las clases Record tradicionales se intercalan las especificaciones de formato de datos con la lógica de empresa de la aplicación. Añadir, cambiar o suprimir un campo significa tener que editar y volver a compilar el código Java. Sin embargo, al utilizar RFML se aíslan las especificaciones de formato de datos en los archivos fuente RFML que están completamente aparte de la lógica de empresa. Acomodar un campo significa únicamente modificar el archivo RFML, a menudo sin tener que cambiar ni volver a compilar la aplicación Java.

En el ejemplo se supone que la aplicación maneja registros de cliente, que ha definido en un archivo fuente RFML y que ha denominado qcustcdt.rfml. El archivo fuente representa los campos que componen cada registro de cliente.

El listado que figura a continuación muestra cómo una aplicación Java puede interpretar un registro de cliente utilizando las clases Record, RecordFormat y FieldDescription de Toolbox para Java:

```
// Almacenamiento intermedio que contiene la representación binaria de un
registro de información.
    byte[] bytes;

    // ... Lea los datos del registro en el almacenamiento intermedio...

    // Configure un objeto RecordFormat para representar un registro de
cliente.
    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"));

    // Lea el almacenamiento intermedio de bytes en el objeto
RecordFormatDocument.
    Record rec1 = new Record(recFmt1, bytes);

    // Obtenga los valores de los campos.
    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"));
```

Por comparación, aquí se muestra cómo se interpretaría el mismo registro utilizando RFML.

El código Java para interpretar el contenido del registro de datos de cliente utilizando RFML podría tener el siguiente aspecto:

```
    // Almacenamiento intermedio que contiene la representación binaria de
un registro de información.
    byte[] bytes;

    // ... Lea los datos del registro en el almacenamiento intermedio...

    // Analice el archivo RFML en un objeto RecordFormatDocument.
    // El archivo fuente RFML se denomina qcustcdt.rfml.
    RecordFormatDocument rfml1 = new RecordFormatDocument("qcustcdt");

    // Lea el almacenamiento intermedio de bytes en el objeto
RecordFormatDocument.
    rfml1.setValues("cusrec", bytes);

    // Obtenga los valores de los campos.
    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"));«
```

Ejemplo: archivo fuente RFML

Este archivo fuente RFML de ejemplo define el formato de los registros de cliente según el uso empleado en el ejemplo de RFML acerca de cómo se utiliza RFML en comparación con el uso de las clases Record de Toolbox para Java. Este archivo fuente RFML sería un archivo de texto denominado qcustcdt.rfml.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
<?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>
```


Clase RecordFormatDocument

La clase RecordFormatDocument permite a los programas Java efectuar la conversión entre representaciones de datos RFML y objetos Record y RecordFormat para su uso con otros componentes de Toolbox para Java.

La clase RecordFormatDocument representa un archivo fuente RFML y proporciona métodos que permiten al programa Java llevar a cabo las acciones siguientes:

- Componer archivos fuente RFML a partir de objetos Record, objetos RecordFormat y matrices de bytes
- Generar objetos Record, objetos RecordFormat y matrices de bytes que representan la información que contiene el objeto RecordFormatDocument
- Obtener y establecer los valores de distintos objetos y tipos de datos
- Generar XML (RFML) que representa los datos que contiene el objeto RecordFormatDocument
- Serializar el archivo fuente RFML que representa el objeto RecordFormatDocument

Para obtener más información sobre los métodos disponibles, consulte el resumen de métodos de javadoc de la clase RecordFormatDocument.

Utilización de la clase RecordFormatDocument con otras clases de Toolbox para Java

Utilice la clase RecordFormatDocument con las siguientes clases de Toolbox para Java:

- Las clases orientadas a registros, que incluyen las clases de archivo de acceso a nivel de registro (AS400File, SequentialFile y KeyedFile) que leen, manipulan y escriben objetos Record. Esta categoría también incluye la clase LineDataRecordWriter.
- Las clases orientadas a bytes, que incluyen determinadas clases DataQueue, UserSpace e IFSFile que leen y escriben una matriz de bytes de datos cada vez.

No utilice la clase RecordFormatDocument con las siguientes clases de Toolbox para Java, que leen y escriben datos en formatos que RecordFormatDocument no maneja:

- Las clases DataArea, ya que los métodos de lectura y escritura sólo manejan los tipos de datos String, boolean y BigDecimal.
- IFSTextFileInputStream e IFSTextFileOutputStream, ya que estos métodos de lectura y escritura sólo manejan el tipo de datos String.
- Las clases JDBC, ya que RFML se centra únicamente en los datos descritos por la especificación de descripción de datos (DDS) de iSeries.

»Documentos RFML y sintaxis RFML

Los documentos RFML, denominados archivos fuente RFML, contienen códigos que definen la especificación de un formato de datos determinado.

Como RFML se basa en PCML, la sintaxis resulta conocida a los usuarios de PCML. Como RFML es una extensión de XML, los archivos fuente RFML son fáciles de leer y sencillos de crear. Por ejemplo, puede crear un archivo fuente RFML mediante un simple editor de texto. Además, los archivos fuente RFML presentan la estructura de los datos de un modo que resulta más sencilla de entender que en un lenguaje de programación como Java.

El ejemplo de RFML Cómo se utiliza RFML en comparación con el uso de las clases Record de Toolbox para Java contiene un archivo fuente RFML de muestra.

DTD RFML

La definición de tipo de documento (DTD) RFML define la sintaxis y los elementos RFML válidos. Para asegurarse de que un analizador XML pueda validar el archivo fuente RFML en el momento de la ejecución, declare la DTD RFML en el archivo fuente:

```
<!DOCTYPE rfml SYSTEM "rfml.dtd">
```

La DTD RFML reside en el archivo jt400.jar (com/ibm/as400/data/rfml.dtd).

Sintaxis de RFML

La DTD RFML define códigos, cada uno de ellos con sus propios códigos de atributos. Utilice los códigos RFML para declarar y definir los elementos siguientes en los archivos fuente RFML:

- El código rfml empieza y termina el archivo fuente RFML que describe el formato de datos.
- El código struct define una estructura con nombre que puede reutilizar en el archivo fuente RFML. La estructura contiene un código de datos para cada uno de los campos de la estructura.
- El código recordformat define un formato de registro, que contiene elementos de datos o referencias a elementos de estructura.
- El código data define un campo dentro de un formato de registro o una estructura.

En el ejemplo siguiente, la sintaxis de RFML describe un formato de registro y una estructura:

```
<rfml>

    <recordformat>
    <data> </data>
    </recordformat>

    <struct>
    <data> </data>
      </struct>

</rfml>
```

«

»

Definición de tipo de documento RFML

Esta es la DTD RFML. Observe que la versión es 4.0. La DTD RFML reside en el archivo jt400.jar (com/ibm/as400/data/rfml.dtd).

```
<!--
Definición de tipo de documento RFML (Record Format Markup Language).

RFML es un lenguaje XML. El uso habitual es:
   <?xml version="1.0"?>
   <!DOCTYPE rfml SYSTEM "rfml.dtd">
   <rfml version="4.0">
   ...
   </rfml>


(C) Copyright IBM Corporation, 2001,2002
Reservados todos los derechos. Material con licencia propiedad de IBM
Derechos restringidos de los usuarios del Gobierno de Estados Unidos
El uso, la duplicación o la divulgación están restringidos
por el GSA ADP Schedule Contract con IBM Corp.
-->

<!-- Entidades de conveniencia -->
<!ENTITY % string             "CDATA">    <!-- una serie de longitud 0 o
superior -->
<!ENTITY % nonNegativeInteger "CDATA">    <!-- un entero no negativo -->
<!ENTITY % binary2            "CDATA">    <!-- un entero del rango 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)">


<!-- El elemento raíz del documento -->
<!ELEMENT rfml (struct | recordformat)+>
<!ATTLIST rfml
          version       %string;    #FIXED "4.0"
          ccsid         %binary2;   #IMPLIED
>
<!-- Nota: el ccsid es el valor por omisión que se utilizará para los
elementos <data type="char"> incluidos que no especifiquen un ccsid. -->


<!-- Nota: RFML no soporta declaraciones struct anidadas. Todos los
elementos struct son hijos directos del nodo raíz. -->
<!ELEMENT struct (data)+>
<!ATTLIST struct
          name          ID          #REQUIRED
>


<!-- <!ELEMENT recordformat (data | struct)*> -->
<!ELEMENT recordformat (data)*>
```

```
<!ATTLIST recordformat
          name          ID                    #REQUIRED
          description   %string;              #IMPLIED
>
<!-- Nota: en el servidor, el campo "text description" de Record sólo puede
tener 50 bytes de longitud. -->

<!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
>
<!-- Nota: el atributo 'name' debe ser exclusivo dentro de un recordformat
determinado. -->
<!-- Nota: en el servidor, los nombres del campo Record sólo pueden tener 10
bytes de longitud. -->
<!-- Nota: el atributo 'length' es obligatorio, excepto si se especifica
type="struct". -->
<!-- Nota: si type="struct", el atributo 'struct' es obligatorio. -->
<!-- Nota: los atributos 'ccsid' y 'bidistringtype' sólo son válidos si se
especifica type="char". -->
<!-- Nota: el atributo 'precision' sólo es válido para los tipos "int",
"packed" y "zoned". -->


<!-- Entidades de caracteres predefinidas estándar -->
<!ENTITY quot  "&#34;">     <!-- comillas -->
<!ENTITY amp   "&#38;#38;"> <!-- signo de unión -->
<!ENTITY apos  "&#39;">     <!-- apóstrofo -->
<!ENTITY lt    "&#38;#60;"> <!-- menos que -->
<!ENTITY gt    "&#62;">     <!-- más que -->
<!ENTITY nbsp  " ">    <!-- espacio de no separación -->
<!ENTITY shy   "&#173;">    <!-- guión virtual (guión discrecional) -->
<!ENTITY mdash "&#38;#x2014;">
<!ENTITY ldquo "&#38;#x201C;">
<!ENTITY rdquo "&#38;#x201D;">
```

Código RFML data

El código data puede tener los atributos que se indican más abajo. Los atributos delimitados por corchetes, [], son opcionales. Si especifica un atributo opcional, no incluya los corchetes en el fuente. A continuación se facilita una lista con algunos valores de atributos delimitados por llaves, {}, y las opciones posibles separadas por barras verticales, |. Cuando especifique uno de estos atributos, no incluya las llaves en el fuente y especifique únicamente una de las opciones que se muestran.

```
<data type="{ char | int | packed | zoned | float | byte | struct }" ]
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 |
DEFAULT }"]
    [ ccsid="{ número | nombre-datos }" ]
    [ count="{ número | nombre-datos }" ]
    [ init="serie" ]
    [ length="{ número | nombre-datos }" ]
    [ name="nombre" ]
    [ precision="número" ]
    [ struct="nombre-estructura" ]>
</data>
```

En la siguiente tabla figuran los atributos del código data. Cada entrada contiene el nombre de atributo, los valores válidos posibles y una descripción del atributo.

Atributo	Valor	Descripción
type=	*char* Un valor de tipo carácter. Un valor de datos de tipo *char* se devuelve como *java.lang.String*.Para obtener más información, consulte los valores *char* para la longitud. *int* Un valor entero. Un valor de datos *int* se devuelve como *java.lang.Long*.Para obtener más información, consulte los valores *int* para la longitud y la precisión. *packed* Un valor decimal empaquetado. Un valor de datos *packed* se devuelve como *java.math.BigDecimal*. Para obtener más información, consulte los valores *packed* para la longitud y la precisión. *zoned* Un valor decimal con zona. Un valor de datos *zoned* se devuelve como *java.math.BigDecimal*.Para obtener más información, consulte los valores *zoned* para la longitud y la precisión. *float* Un valor de coma flotante. El atributo **length** especifica el número de bytes (4 o 8). Un entero de 4 bytes se devuelve como *java.lang.Float*. Un entero de 8 bytes se devuelve como *java.lang.Double*. Para obtener más información, consulte los valores *float* para la longitud. *byte* Un valor de tipo byte. No se efectúa ninguna conversión de los datos. Un valor de datos *byte* se devuelve como una matriz de valores *byte* (*byte[]*). Para obtener más	Indica el tipo de datos que se utiliza (carácter, entero, empaquetado, con zona, coma flotante o estructura). Los valores de los atributos de longitud y precisión varían según los diferentes tipos de datos. Para obtener más información, consulte los valores de longitud y precisión.

	información, consulte los valores *byte* para la longitud. *struct* El nombre del elemento **<struct>**. Un valor *struct* permite definir una estructura una vez y volver a utilizarla varias veces dentro del mismo documento. Si utiliza **type=**"`struct`", es como si la estructura especificada apareciera en ese lugar del documento. Un valor *struct* no permite un valor de longitud y no tiene ningún valor para la precisión.	
bidistringtype=	*DEFAULT* donde DEFAULT es el tipo serie por omisión para datos no bidireccionales (LTR). *ST4* donde *ST4* es el tipo de serie 4. *ST5* donde *ST5* es el tipo de serie 5. *ST6* donde *ST6* es el tipo de serie 6. *ST7* donde *ST7* es el tipo de serie 7. *ST8* donde *ST8* es el tipo de serie 8. *ST9* donde *ST9* es el tipo de serie 9. *ST10* donde *ST10* es el tipo de serie 10. *ST11* donde *ST11* es el tipo de serie 11.	Especifica el tipo de serie bidireccional para los elementos **<data>** que tengan **type=**"`char`". Si se omite este atributo, el tipo de serie para este elemento viene determinado por el CCSID (especificado explícitamente o el CCSID por omisión del entorno de sistema principal). Los tipos de serie están definidos en el javadoc correspondiente a la clase BidiStringType.
ccsid=	*número* donde *número* define un CCSID fijo, que no cambia nunca. *nombre-datos* donde *nombre-datos* define el nombre que contendrá, en tiempo de ejecución, el CCSID de los datos de tipo carácter. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento **<data>** que se haya definido con **type=**"`int`". Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica el CCSID (ID de juego de caracteres codificados) de sistema principal de los datos de tipo carácter para el elemento **<data>**. El atributo **ccsid** sólo se puede especificar para los elementos **<data>** que tengan **type=**"`char`". Si se omite este atributo, se presupone que los datos de tipo carácter de este elemento tienen el CCSID por omisión del entorno de sistema principal.

count=	*número* donde *número* define un número fijo de elementos de una matriz dimensionada. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento RFML que contendrá, en tiempo de ejecución, el número de elementos de la matriz. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con **type**=`"int"`. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica que el elemento es una matriz e identifica el número de entradas de la matriz. Si se omite el atributo `count`, el elemento no está definido como matriz, aunque puede encontrarse dentro de otro elemento que esté definido como matriz.
init=	*serie*	Especifica un valor inicial para el elemento <**data**>. El valor inicial especificado se utiliza para inicializar los valores escalares. Si el elemento está definido como matriz o se encuentra dentro de una estructura definida como matriz, el valor inicial especificado se utiliza como valor inicial para todas las entradas de la matriz.
length=	*número* donde *número* define una longitud fija. *nombre-datos* donde *nombre-datos* define el nombre de un elemento <**data**> dentro del documento RFML que contendrá, en tiempo de ejecución, la longitud. Únicamente puede especificarse un *nombre-datos* para los elementos <**data**> que tengan **type**=`"char"` o **type**=`"byte"`. El *nombre-datos* especificado puede ser un nombre totalmente calificado o un nombre relativo al elemento actual. En los dos casos, el nombre debe hacer referencia a un elemento <**data**> que se haya definido con **type**=`"int"`. Si desea más información sobre cómo se resuelven los nombres relativos, consulte el apartado Resolución de nombres relativos.	Especifica la longitud del elemento de datos. El uso de este atributo varía en función del tipo de datos. Para obtener más información, consulte los valores de longitud y precisión.
name=	*nombre*	Especifica el nombre del elemento <**data**>.
precision=	*número*	Especifica el número de bytes de precisión para algunos tipos de datos numéricos. Para obtener más información, consulte los valores de longitud y precisión.
struct=	*nombre*	Especifica el nombre de un elemento <**struct**> para el elemento <**data**>. Un atributo **struct** sólo se puede especificar para los elementos <**data**> que tengan **type**=`"struct"`.

»Código RFML rfml

El código rfml puede tener los atributos que se indican más abajo. Los atributos delimitados por corchetes, [], son opcionales. Si especifica un atributo opcional, no incluya los corchetes en el fuente.

```
<rfml version="serie-versión"
    [ ccsid="número" ]>
</rfml>
```

En la siguiente tabla figuran los atributos del código rfml. Cada entrada contiene el nombre de atributo, los valores válidos posibles y una descripción del atributo.

Atributo	Valor	Descripción
version=	*serie-versión* Una versión fija de la DTD RFML. Para la versión V5R2, `4.0` es el único valor válido.	Especifica la versión de la DTD RFML, que puede emplear para verificar el valor correcto.
ccsid=	*número* Un CCSID (ID de juego de caracteres codificados) fijo e invariable.	Especifica el CCSID de sistema principal, que es válido para todas los elementos <**data type**="char"> incluidos que no especifiquen ningún CCSID. Para obtener más información, consulte el código **<data>** RFML. . Si omite este atributo, se utilizará el CCSID por omisión del entorno del sistema principal.

«

»Código RFML recordformat

El código recordformat puede tener los atributos que se indican más abajo. Los atributos delimitados por corchetes, [], son opcionales. Si especifica un atributo opcional, no incluya los corchetes en el fuente.

```
<recordformat name="nombre"
    [ description="descripción" ]>
</recordformat>
```

En la siguiente tabla figuran los atributos del código recordformat. Cada entrada contiene el nombre de atributo, los valores válidos posibles y una descripción del atributo.

Atributo	Valor	Descripción
name=	*nombre*	Especifica el nombre del formato de registro.
description=	*descripción*	Especifica la descripción del formato de registro.

«

»Código RFML struct

El código struct puede tener los atributos que se indican más abajo. Los atributos delimitados por corchetes, [], son opcionales. Si especifica un atributo opcional, no incluya los corchetes en el fuente.

```
<struct name="nombre">
</struct>
```

En la siguiente tabla figuran los atributos del código struct. Cada entrada contiene el nombre de atributo, los valores válidos posibles y una descripción del atributo.

Atributo	Valor	Descripción
name=	*nombre*	Especifica el nombre del elemento <**struct**>.

«

SSL (capa de sockets segura) y Java Secure Socket Extension

IBM Toolbox para Java soporta el uso de Java Secure Socket Extension (JSSE) para las conexiones SSL (capa de sockets segura). JSSE está disponible como paquete opcional de la plataforma Java 2, Standard Edition (J2SE), versiones 1.2 y 1.3. JSSE está integrado en J2SE, versión 1.4.

Para obtener más información sobre JSSE, consulte el sitio Web de JSSE de Sun.

JSSE ofrece la posibilidad de llevar a cabo la autenticación de servidor, permitir las comunicaciones seguras y cifrar datos. Con JSSE, puede hacer posible un intercambio de datos seguro entre los clientes y servidores que ejecutan cualquier protocolo de aplicaciones (por ejemplo, HTTP y FTP) a través de TCP/IP.

Una vez instalado y configurado JSSE, Toolbox para Java lo utiliza por omisión. Puede migrar a JSSE dado que ya no se efectuarán mejoras en sslight. La información acerca de cómo utilizar sslight para habilitar SSL sólo se facilita a efectos de compatibilidad con versiones anteriores.

Antes de empezar a utilizar SSL con IBM Toolbox para Java, debe saber cuáles son sus responsabilidades legales.

Depurador del sistema iSeries

IBM iSeries System Debugger proporciona un nuevo entorno gráfico de depuración para el usuario en el servidor iSeries. Utilice iSeries System Debugger para depurar y probar los programas que se ejecutan en el servidor iSeries, incluidos aquellos que se ejecutan en el entorno PASE de OS/400.

Encontrará más información acerca de iSeries System Debugger en los siguientes temas:

Componentes
Infórmese de los distintos componentes de que consta iSeries System Debugger y de cómo trabajan conjuntamente a fin de proporcionar una avanzada herramienta de depuración.

Instalación
Descubra cuáles son los requisitos de instalación de iSeries System Debugger y aprenda a instalarlo en la estación de trabajo.

Ejecución de iSeries System Debugger
Consulte cómo puede ejecutar los distintos componentes de depuración.

Componentes del depurador del sistema iSeries

El depurador del sistema iSeries consta de los siguientes componentes:

- Un gestor de depuración basado en el cliente
- Un depurador del sistema basado en el cliente
- Un depurador OS/400 PASE basado en el cliente
- Un concentrador de depuración basado en el sistema principal
- Un servidor de depuración basado en el sistema principal

Las descripciones siguientes proporcionan únicamente información general sobre los componentes del depurador del sistema iSeries. Para obtener más información sobre los componentes, ejecute el depurador del sistema iSeries y consulte la ayuda en línea. Para ver la ayuda en línea del depurador del sistema iSeries, lleve a cabo una de las acciones siguientes:

- En el menú **Ayuda** de cualquier interfaz del depurador, pulse **Ayuda**.
- Pulse **F1**.

Gestor de depuración

El gestor de depuración registra el cliente en el concentrador de depuración, que permite utilizar la modalidad de depuración gráfica para los sistemas seleccionados. Una vez registrado, cuando un cliente emite el mandato CL Iniciar depuración (STRDBG) desde una sesión de emulación se inicia el depurador del sistema.

Utilice el gestor de depuración para gestionar las conexiones y operaciones de depuración:

- Añadir y eliminar sistemas
- Añadir y eliminar usuarios
- Iniciar operaciones de depuración
- Lanzar el depurador del sistema

Depurador del sistema

Utilice el depurador del sistema iSeries para depurar los programas que se ejecutan en el servidor iSeries. Puede depurar los programas que se ejecutan en trabajos existentes en el sistema o utilizar el depurador del sistema para lanzar y a continuación depurar programas en un trabajo de proceso por lotes del sistema.

Puede configurar el depurador del sistema para que se inicie automáticamente, manualmente desde una solicitud de mandatos de estación de trabajo o mediante la interfaz del gestor de depuración.

Utilice el depurador del sistema para llevar a cabo actividades de depuración tales como:

- Establecer puntos de interrupción
- Recorrer programas
- Inspeccionar variables
- Examinar la pila de llamadas
- Examinar la memoria asociada a las variables de programa
- Examinar la actividad de las hebras

Depurador OS/400 PASE

Utilice el depurador OS/400 PASE para depurar programas que se ejecutan en un entorno OS/400 PASE. Puede depurar programas que se ejecutan en unproceso existente en el sistema o utilizar el depurador OS/400 PASE para lanzar un programa y después depurarlo.

Puede lanzar el depurador OS/400 PASE directamente desde la línea de mandatos o mediante la interfaz gestor de depuración.

Además de las actividades de depuración enumeradas anteriormente para el depurador del sistema, puede utilizar el depurador OS/400 PASE para realizar actividades de depuración específicas de OS/400 PASE, que son las siguientes:

- Utilización del mapa de carga del programa a depurar
- Visualización de una lista de archivos fuente y métodos
- Seguimiento de los procesos padres e hijos
- Examen de registros

Concentrador de depuración

El concentrador de depuración proporciona las funciones siguientes:

- Sirve como registro para los clientes que desean utilizar el depurador del sistema »o el depurador OS/400 PASE.«
- Maneja las peticiones entrantes de inicio de servidores de depuración.

Utilice la interfaz del gestor de depuración para registrar el cliente en el concentrador de depuración. Al registrar un cliente se almacena la información de usuario y la dirección TCP/IP del cliente en el registro. Al emitir el mandato CL Iniciar depuración (STRDBG) desde una sesión de emulación se establece contacto con el concentrador de depuración para ver si el usuario que ejecuta el mandato está registrado en el gestor de depuración. Asimismo, comprueba si el mandato que se ejecuta procede de la misma dirección TCP/IP que el gestor de depuración. Si se cumplen estas condiciones, se inicia la aplicación gráfica del depurador del sistema iSeries en lugar del entorno de depuración tradicional.

El concentrador de depuración también sirve de único punto de contacto para todas las aplicaciones de depuración del sistema iSeries. Cuando un »componente « del depurador del sistema iSeries lleva a cabo una operación de inicio de depuración, el concentrador de depuración somete un trabajo servidor de depuración en nombre del usuario y le pasa la conexión TCP/IP asociada.

Servidor de depuración

El servidor de depuración es un servidor TCP/IP que el concentrador de depuración inicia cuando »uno de los depuradores« emite una petición de inicio de depuración. A continuación, el trabajo servidor da servicio al trabajo que se depura y emite las API y los mandatos del depurador adecuados.«

Instalación del depurador del sistema iSeries

Para ejecutar el depurador del sistema iSeries, la estación de trabajo cliente debe cumplir los siguientes requisitos de hardware y software.

Requisitos de hardware

Debe tener el siguiente hardware instalado en el cliente:

- CPU: 400-500 MHz
- Memoria: 128 MB mínimo (256 MB recomendado)

Requisitos de software

Debe tener el siguiente software instalado en el cliente:

- Uno de los elementos siguientes:
  - Plataforma Java 2, Standard Edition (J2SE) o Enterprise Edition (J2EE), versión 1.3 o posterior
  - Java 2 Runtime Environment (JRE), Standard Edition, versión 1.3.1 o posterior
- jhall.jar (uno de los archivos jar de JavaHelp(TM))

Nota: no olvide añadir jhall.jar a la variable de entorno CLASSPATH del cliente.

Para obtener información sobre cómo instalar el software indicado anteriormente, consulte el sitio Web de Java de Sun.

Instalación del archivo jar del depurador del sistema iSeries

El depurador del sistema iSeries se incluye dentro del paquete de IBM Toolbox para Java. Si todavía no tiene instalado el archivo jt400.jar de Toolbox para Java en el cliente, tendrá que instalarlo cuando instale el depurador del sistema iSeries.

Antes de instalar el depurador del sistema iSeries, compruebe que el sistema cliente cumple los requisitos indicados anteriormente. Para instalar el depurador del sistema iSeries, siga estos pasos:

1. Instale Toolbox para Java y asegúrese de que copia jt400.jar y tes.jar en el cliente.

   Nota: si ya ha instalado Toolbox para Java en el servidor, los archivos jt400.jar y tes.jar estarán en el mismo directorio del servidor:

   `/QIBM/ProdData/HTTP/Public/jt400/lib/`
2. Tras copiar los archivos jar en el cliente, añádalos a la variable de entorno CLASSPATH del cliente.

Ahora puede emplear el cliente para ejecutar el depurador del sistema iSeries.

»Ejecución del depurador del sistema iSeries

Puede utilizar una solicitud de mandatos de cliente para iniciar el gestor de depuración, iniciar el depurador del sistema o iniciar el depurador OS/400 PASE.

Para obtener más información sobre el depurador del sistema iSeries, inicie el depurador del sistema iSeries y consulte la ayuda en línea. Para ver la ayuda en línea del depurador del sistema iSeries, lleve a cabo una de las acciones siguientes:

- En el menú **Ayuda** de cualquier interfaz del depurador, pulse **Ayuda**.
- Pulse **F1**.

Inicio del gestor de depuración

Para iniciar el gestor de depuración desde una solicitud de mandatos del cliente, ejecute el mandato siguiente:

```
java utilities.DebugMgr
```

Inicio del depurador del sistema

Para iniciar el depurador del sistema desde una solicitud de mandatos del cliente, ejecute el mandato siguiente:

```
java utilities.Debug <args>
```

donde `<args>` representa cualquiera de los siguientes argumentos del mandato:

- `-u` = usuario
- `-s` = nombre de sistema
- `-j` = descripción de trabajo, con el formato número de trabajo/usuario de trabajo/nombre de trabajo
- `-p` = programa que debe ejecutarse, con el formato biblioteca de programa/nombre de programa

Nota: una vez que utilice el gestor de depuración para registrar el cliente, puede emitir el mandato CL Iniciar depuración (STRDBG) desde una sesión de emulación para iniciar el depurador del sistema. También puede lanzar el depurador del sistema directamente desde el gestor de depuración del sistema.

Inicio del depurador OS/400 PASE

Para iniciar el depurador OS/400 PASE desde una solicitud de mandatos del cliente, ejecute el mandato siguiente:

```
java utilities.DebugPASE <args>
```

donde `<args>` representa cualquiera de los siguientes argumentos del mandato:

- `-u` = usuario
- `-s` = nombre de sistema
- `-p` = vía de acceso totalmente calificado del programa que debe ejecutarse
- `-pid` = ID de proceso

Nota: puede lanzar el depurador OS/400 PASE directamente desde el gestor de depuración del sistema. A diferencia del depurador del sistema, no puede lanzar el depurador OS/400 PASE desde una sesión de emulación.

Para obtener más información sobre el depurador del sistema iSeries, ejecute el depurador del sistema iSeries y consulte la ayuda en línea. Para ver la ayuda en línea del depurador del sistema iSeries:

- En el menú **Ayuda** de cualquier interfaz del depurador, pulse **Ayuda**.
- Desde cualquier interfaz del depurador del sistema iSeries, pulse **F1**.«

Ejemplo: archivo de propiedades

```
#============================================================#
# IBM Toolbox para Java                                      #
#------------------------------------------------------------#
# Archivo de propiedades de ejemplo                          #
#                                                            #
# Denomine este archivo jt400.properties y almacénelo en     #
# un directorio com/ibm/as400/access al que apunte la        #
# sentencia CLASSPATH.                                       #
#============================================================#

#------------------------------------------------------------#
# Propiedades del sistema de servidor proxy                  #
#------------------------------------------------------------#

# Esta propiedad del sistema especifica el nombre de sistema principal y el
# número de puerto del servidor proxy, con el formato:
nombresisprincipal:númeropuerto
# El número de puerto es opcional.
com.ibm.as400.access.AS400.proxyServer=hqoffice

# Esta propiedad del sistema especifica qué parte del flujo de datos de
proxy
# se cifra mediante SSL. Los valores válidos son:
#   1 - De cliente proxy a servidor proxy
#   2 - De servidor proxy a AS/400
#   3 - De cliente proxy a proxy y de servidor proxy a AS/400
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=1

# Esta propiedad del sistema especifica con qué frecuencia, en segundos,
# el servidor proxy buscará si hay conexiones desocupadas. El
# servidor proxy inicia una hebra para buscar los clientes que ya no
# tienen comunicación. Utilice esta propiedad para establecer con qué
# frecuencia busca la hebra si hay conexiones desocupadas.
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval=7200

# Esta propiedad del sistema especifica durante cuánto tiempo, en segundos,
un
# cliente puede estar desocupado antes de eliminarse. El servidor proxy
# inicia una hebra para buscar los clientes que ya no tienen comunicación.
# Utilice esta propiedad para establecer durante cuánto tiempo puede estar
# desocupado un cliente antes de eliminarse.
com.ibm.as400.access.TunnelProxyServer.clientLifetime=2700


#------------------------------------------------------------#
# Propiedades del sistema de rastreo                         #
#------------------------------------------------------------#

# Esta propiedad del sistema especifica qué categorías de rastreo deben
habilitarse.
# Se trata de una lista delimitada por comas que contiene cualquier
combinación de
# categorías de rastreo. La lista completa de categorías de rastreo se
define en
# la clase Trace.
com.ibm.as400.access.Trace.category=error,warning,information
```

```
# Esta propiedad del sistema especifica el archivo en que se escribe la
# salida de rastreo. Por omisión la salida de rastreo se escribe en
System.out.
com.ibm.as400.access.Trace.file=c:\\temp\\trace.out


#-------------------------------------------------------------#
# Propiedades del sistema de llamada a mandato                #
#-------------------------------------------------------------#

# Esta propiedad del sistema especifica si las llamadas a mandato deben
# suponerse como seguras en ejecución multihebra. Si es true, se supone que
# todas las llamadas a mandato son seguras en ejecución multihebra. Si es
false,
# se supone que todas las llamadas a mandato no son seguras en ejecución
multihebra.
# Esta propiedad se omite para un objeto CommandCall determinado si
# CommandCall.setThreadSafe(true/false) o
# AS400.setMustUseSockets(true) se ha llevado a cabo sobre el objeto.
com.ibm.as400.access.CommandCall.threadSafe=true


#-------------------------------------------------------------#
# Propiedades del sistema de llamada a programa               #
#-------------------------------------------------------------#

# Esta propiedad del sistema especifica si las llamadas a programa deben
# suponerse como seguras en ejecución multihebra. Si es true, se supone que
# todas las llamadas a programa son seguras en ejecución multihebra. Si es
false,
# se supone que todas las llamadas a programa no son seguras en ejecución
multihebra.
# Esta propiedad se omite para un objeto ProgramCall determinado si
# ProgramCall.setThreadSafe(true/false) o
# AS400.setMustUseSockets(true) se ha llevado a cabo sobre el objeto.
com.ibm.as400.access.ProgramCall.threadSafe=true


# End
```

Ejemplo: archivo fuente de clase de propiedades del sistema

```
//=============================================================
// IBM Toolbox para Java
//-------------------------------------------------------------
// Archivo fuente de clase de propiedades de ejemplo
//
// Compile este archivo fuente y almacene el archivo de
// clase en la sentencia CLASSPATH.
//=============================================================
package com.ibm.as400.access;

public class Properties
extends java.util.Properties
{
      public Properties ()
   {
      /*-----------------------------------------------------------*/
      /* Propiedades del sistema de servidor proxy                 */
      /*-----------------------------------------------------------*/

      // Esta propiedad del sistema especifica el nombre de sistema
principal y el
      // número de puerto del servidor proxy, con el formato:
nombresisprincipal:númeropuerto
      // El número de puerto es opcional.
         put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");

      // Esta propiedad del sistema especifica qué parte del flujo de datos
de proxy
      // se cifra mediante SSL. Los valores válidos son:
      //   1 - De cliente proxy a servidor proxy
      //   2 - De servidor proxy a servidor iSeries o AS/400e
      //   3 - De cliente proxy a proxy y de servidor proxy a servidor
iSeries o AS/400e
      put("com.ibm.as400.access.SecureAS400.proxyEncryptionMode", "1");

      // Esta propiedad del sistema especifica con qué frecuencia, en
segundos,
      // el servidor proxy buscará si hay conexiones desocupadas. El
      // servidor proxy inicia una hebra para buscar los clientes que ya no
      // tienen comunicación. Utilice esta propiedad para establecer con qué
      // frecuencia busca la hebra si hay conexiones desocupadas.
      put("com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval",
"7200");

      // Esta propiedad del sistema especifica durante cuánto tiempo, en
segundos, un
      // cliente puede estar desocupado antes de eliminarse. El servidor
proxy
      // inicia una hebra para buscar los clientes que ya no tienen
comunicación.
      // Utilice esta propiedad para establecer durante cuánto tiempo puede
estar
      // desocupado un cliente antes de eliminarse.
      put("com.ibm.as400.access.TunnelProxyServer.clientLifetime", "2700");
```

```
      /*-----------------------------------------------------------*/
      /* Propiedades del sistema de rastreo                        */
      /*-----------------------------------------------------------*/

      // Esta propiedad del sistema especifica qué categorías de rastreo
deben habilitarse.
      // Se trata de una lista delimitada por comas que contiene cualquier
combinación de
      // categorías de rastreo. La lista completa de categorías de rastreo
se define en
      // la clase Trace.
      put ("com.ibm.as400.access.Trace.category",
"error,warning,information");

      // Esta propiedad del sistema especifica el archivo en que se escribe
la
      // salida de rastreo. Por omisión la salida de rastreo se escribe en
System.out.
      put ("com.ibm.as400.access.Trace.file", "c:\temp\trace.out");


      /*-----------------------------------------------------------*/
      /* Propiedades del sistema de llamada a mandato              */
      /*-----------------------------------------------------------*/

      // Esta propiedad del sistema especifica si las llamadas a mandato
deben
      // suponerse como seguras en ejecución multihebra. Si es true, se
supone que
      // todas las llamadas a mandato son seguras en ejecución multihebra.
Si es false,
      // se supone que todas las llamadas a mandato no son seguras en
ejecución multihebra.
      // Esta propiedad se omite para un objeto CommandCall determinado si
      // CommandCall.setThreadSafe(true/false) o
      // AS400.setMustUseSockets(true) se ha llevado a cabo sobre el objeto.
      put ("com.ibm.as400.access.CommandCall.threadSafe", "true");


      /*-----------------------------------------------------------*/
      /* Propiedades del sistema de llamada a programa             */
      /*-----------------------------------------------------------*/

      // Esta propiedad del sistema especifica si las llamadas a programa
deben
      // suponerse como seguras en ejecución multihebra. Si es true, se
supone que
      // todas las llamadas a programa son seguras en ejecución multihebra.
Si es false,
      // se supone que todas las llamadas a programa no son seguras en
ejecución multihebra.
      // Esta propiedad se omite para un objeto ProgramCall determinado si
      // ProgramCall.setThreadSafe(true/false) o
      // AS400.setMustUseSockets(true) se ha llevado a cabo sobre el objeto.
      put ("com.ibm.as400.access.ProgramCall.threadSafe", "true");
   }
}
```

»IBM Toolbox para Java 2 Micro Edition

El paquete IBM Toolbox para Java 2 Micro Edition (com.ibm.as400.micro) permite escribir programas Java con los que diversos dispositivos inalámbricos Tier0, como los organizadores personales (PDA) y los teléfonos móviles, puedan acceder directamente a los datos y recursos del iSeries.

Encontrará más información acerca de ToolboxME para iSeries en estos temas:

Requisitos
Obtenga información sobre los requisitos que deben cumplirse para desarrollar aplicaciones con ToolboxME para iSeries y ejecutar esas aplicaciones en los dispositivos Tier0.

Bajar y configurar ToolboxME para iSeries
Aprenda a bajar y configurar ToolboxME para iSeries en el servidor, la estación de trabajo y el dispositivo Tier0.

Conceptos
Lea una breve introducción en la que se definen los conceptos importantes para el desarrollo de aplicaciones que se ejecutan en dispositivos Tier0.

Clases de ToolboxME para iSeries
Infórmese acerca de las clases del componente ToolboxME para iSeries (paquete com.ibm.as400.micro). Las clases proporcionan un conjunto reducido de las funciones disponibles en las clases de Toolbox para Java, soporte para JDBC y mucho más.

Crear un programa de ToolboxME para iSeries
Aprenda el procedimiento que debe seguirse para crear programas de ToolboxME para iSeries que se ejecuten en un dispositivo Tier0. Siga las instrucciones y cree su primer programa de ToolboxME para iSeries.

Ejemplos
Examine, baje y ejecute los ejemplos de trabajo de ToolboxMe para iSeries que le ayudarán a entender cómo crear y utilizar aplicaciones inalámbricas.«

Bajar y configurar ToolboxME para iSeries

Debe bajar y configurar ToolboxME para iSeries (jt400Micro.jar), que se encuentra en JTOpen, por separado. Puede bajar ToolboxME para iSeries del sitio Web de IBM Toolbox para Java/JTOpen, que también ofrece información adicional acerca de cómo configurar ToolboxME para iSeries.

El procedimiento de configuración de ToolboxME para iSeries varía para el dispositivo Tier0, la estación de trabajo y el servidor:

- Cree una aplicación para el dispositivo inalámbrico (mediante jt400Micro.jar) e instale la aplicación según las indicaciones del fabricante del dispositivo.
- Compruebe que los servidores de sistema principal iSeries estén iniciados en el servidor que contiene los datos de destino.
- Asegúrese de que el sistema que desea que ejecute MEServer tenga acceso a jt400.jar. Para obtener más información, consulte la información acerca de la instalación de Toolbox para Java en la estación de trabajo y la instalación de Toolbox para Java en un servidor iSeries.

Conceptos importantes para el uso de ToolboxME para iSeries

Antes de empezar a desarrollar aplicaciones Java de ToolboxME para iSeries, debe entender los siguientes conceptos y estándares que rigen ese desarrollo.

Plataforma Java 2, Micro Edition (J2ME)

J2ME(TM) es la implementación del estándar Java 2 que proporciona entornos de ejecución Java para los dispositivos inalámbricos Tier0, tales como organizadores personales (PDA) y teléfonos móviles. IBM Toolbox para Java 2 Micro Edition cumple este estándar.

Dispositivos Tier0

Los dispositivos inalámbricos, como los PDA y los teléfonos móviles, que utilizan la tecnología inalámbrica para conectarse a sistemas y redes se denominan dispositivos Tier0. Este nombre deriva del modelo de aplicación en 3 niveles común. El modelo de 3 niveles describe un programa distribuido que está organizado en tres partes principales, cada una de las cuales reside en un sistema o una red diferente.

- El tercer nivel es la base de datos y los programas relacionados que residen en un servidor, que a menudo es un servidor distinto del del segundo nivel. Este nivel proporciona la información y el acceso a esa información que los demás niveles utilizan para llevar a cabo el trabajo.
- El segundo nivel es el de la lógica de empresa, que normalmente reside en otro sistema (normalmente un servidor) compartido en una red.
- El primer nivel por lo general es la parte de la aplicación que reside en una estación de trabajo, que incluye la interfaz de usuario.

Los dispositivos Tier0 con frecuencia son dispositivos pequeños, fáciles de transportar y con recursos restringidos, como por ejemplo los PDA y los teléfonos móviles. Los dispositivos Tier0 sustituyen o complementan las funciones de los dispositivos del primer nivel.

CLDC (Connected Limited Device Configuration)

Una configuración define un conjunto mínimo de interfaces API y las posibilidades necesarias de una máquina virtual Java para proporcionar las funciones esperadas para un gran conjunto de dispositivos. La CLDC está destinada al amplio conjunto de dispositivos con restricciones de recursos que incluyen los dispositivos Tier0.

Para obtener más información, consulte CLDC and the K Virtual Machine (KVM)  .

MIDP (Mobile Information Device Profile)

Un perfil representa un conjunto de interfaces API creado a partir de una configuración existente destinada a un tipo de dispositivo o sistema operativo específico. El MIDP, creado tomando como base la CLDC, proporciona un entorno de ejecución estándar que permite desplegar dinámicamente aplicaciones y servicios en los dispositivos Tier0.

Para obtener más información, consulte Mobile Information Device Profile (MIDP)  .

Máquina virtual Java para dispositivos inalámbricos

Para ejecutar la aplicación Java, el dispositivo Tier0 requiere una máquina virtual Java diseñada específicamente para los recursos limitados de un dispositivo inalámbrico. Entre las máquinas virtuales Java que puede utilizar se encuentran las siguientes:

- Máquina virtual IBM J9, que forma parte de IBM WebSphere Micro Environment
- Sun K Virtual Machine (KVM)
- MIDP

Información relacionada

Puede emplear cualquiera de las diversas herramientas de desarrollo diseñadas para ayudarle a crear aplicaciones Java inalámbricas. Puede obtener una lista de estas herramientas en la información relacionada acerca de IBM Toolbox para Java.

Para obtener más información al respecto y bajar simuladores y emuladores de dispositivos inalámbricos, consulte en el sitio Web del dispositivo o sistema operativo en el que desea que se ejecute la aplicación.

»Clases de ToolboxME para iSeries

El paquete com.ibm.as400.micro proporciona las clases necesarias para escribir aplicaciones que permitan al dispositivo Tier 0 acceder a los datos y recursos del servidor iSeries.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries.

ToolboxME para iSeries proporciona las clases siguientes:

- MEServer media entre las peticiones del dispositivo Tier0 y el servidor de sistema principal.
- Varias clases proporcionan un subconjunto de las funciones del paquete access de Toolbox para Java.
  - AS400 - Inicia la sesión en un sistema iSeries.
  - CommandCall - Llama a un mandato iSeries.
  - DataQueue - Lee información de una cola de datos del servidor iSeries y escribe información en ella.
  - ProgramCall - Llama a un programa del servidor iSeries y accede a los datos devueltos una vez ejecutado el programa.
- Otras clases proporcionan soporte para JDBC, que también incluye el conjunto mínimo de métodos y datos de utilidad del paquete java.sql.«

Clase MEServer

Utilice la clase MEServer para satisfacer las peticiones de la aplicación cliente Tier0 que utiliza el archivo jar de ToolboxME para iSeries. MEServer crea objetos de Toolbox para Java e invoca métodos sobre ellos en nombre de la aplicación cliente.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más información, consulte Bajar y configurar ToolboxME para iSeries.

Utilice el mandato siguiente para iniciar un MEServer:

```
java com.ibm.as400.micro.MEServer [opciones]
```

donde [opciones] es una o varias de las opciones siguientes:

-pcml pcml-doc1 [;pcml_doc2;...]

Especifica el documento PCML que debe precargarse y analizarse. Puede abreviar esta opción utilizando -pc.

Para obtener información importante acerca de cómo utilizar esta opción, consulte el javadoc de MEServer.

-port puerto

Especifica el puerto que se utilizará para aceptar conexiones de clientes. Esta opción puede abreviarse especificando -po. El puerto por omisión es 3470. Puede abreviar esta opción utilizando -po.

-verbose [true|false]

Especifica si debe imprimirse la información de estado y conexión en System.out. Puede abreviar esta opción utilizando -v.

-help

Imprime información de uso en System.out. Puede abreviar esta opción utilizando -h o -?. El valor por omisión establece no imprimir la información de uso.

MEServer no se iniciará si ya hay otro servidor activo en el puerto especificado.

Clase AS400

La clase AS400 del paquete micro (com.ibm.as400.micro.AS400) proporciona un subconjunto modificado de las funciones disponibles en la clase AS400 del paquete access (com.ibm.as400.access.AS400). Utilice la clase AS400 de ToolboxMe para iSeries a fin de iniciar la sesión en un sistema iSeries desde un dispositivo Tier0.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más información, consulte Bajar y configurar ToolboxME para iSeries.

La clase AS400 proporciona las siguientes funciones:

- Conectarse a MEServer
- Desconectarse de MEServer

La conexión con MEServer se efectúa implícitamente. Por ejemplo, tras crear un objeto AS400, puede emplear el método run() en CommandCall para llevar a cabo automáticamente connect(). Dicho de otro modo, no es necesario que llame explícitamente al método connect() salvo que desee controlar cuándo se establece la conexión.

El ejemplo que sigue muestra cómo se utiliza la clase AS400 para iniciar la sesión en un sistema iSeries:

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
                           try
{
    system.connect();
}
      catch (Exception e)
{
    // Maneje la excepción.
}
// Se ha terminado de trabajar con el objeto del sistema.
system.disconnect();
```

Clase CommandCall

La clase CommandCall del paquete micro (com.ibm.as400.micro.CommandCall) proporciona un subconjunto modificado de las funciones disponibles en la clase CommandCall del paquete access (com.ibm.as400.access.CommandCall). Utilice la clase CommandCall para llamar a un mandato iSeries desde un dispositivo Tier0.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries.

El método run() de CommandCall requiere una serie (el mandato que desea ejecutar) y devuelve los mensajes que se obtienen tras la ejecución del mandato como una serie. Si el mandato se ejecuta pero no genera ningún mensaje, el método run() devuelve una matriz de tipo serie vacía.

El ejemplo siguiente muestra cómo se puede utilizar CommandCall:

```
// Trabaje con mandatos.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
                          try
{
    // Ejecute el mandato "CRTLIB FRED."
    String[] messages = CommandCall.run(system, "CRTLIB FRED");
    if (messages != null)
    {
        // Observe que había un error.
        System.out.println("Mandato anómalo:");
        for (int i = 0; i < messages.length; ++i)
        {
            System.out.println(messages[i]);
        }
    }
       else
    {
        System.out.println("Mandato satisfactorio.");
    }
}
     catch (Exception e)
{
    // Maneje la excepción.
}
// Se ha terminado de trabajar con el objeto del sistema.
system.disconnect();
```

»

Clase DataQueue

La clase DataQueue del paquete micro (com.ibm.as400.micro.DataQueue) proporciona un subconjunto modificado de las funciones disponibles en la clase DataQueue del paquete access (com.ibm.as400.access.DataQueue). Utilice la clase DataQueue para que el dispositivo Tier0 lea información de una cola de datos del servidor iSeries o escriba información en ella.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries.

La clase DataQueue incluye estos métodos:

- Leer o escribir una entrada como una serie
- Leer o escribir una entrada como una matriz de bytes

Para leer o escribir entradas, debe proporcionar el nombre del sistema iSeries donde reside la cola de datos y el nombre de vía de acceso del sistema de archivos integrado de la cola de datos. Cuando no hay entradas disponibles, al leer una entrada se devuelve un valor nulo.

En el ejemplo siguiente se muestra cómo utilizar la clase DataQueue para leer entradas de una cola de datos de un sistema iSeries y escribir entradas en ella:

```
   AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
                             try
   {
       // Escriba información en la cola de datos.
       DataQueue.write(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ", "texto");

       // Lea información de la cola de datos.
       String txt = DataQueue.read(system,
"/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");
   }
        catch (Exception e)
   {
       // Maneje la excepción.
   }
   // Se ha terminado de trabajar con el objeto del sistema.
   system.disconnect();
```

«

Clase ProgramCall

La clase ProgramCall del paquete micro (com.ibm.as400.micro.ProgramCall) proporciona un subconjunto modificado de las funciones disponibles en la clase ProgramCall del paquete access (com.ibm.as400.access.ProgramCall). Utilice la clase ProgramCall para permitir a un dispositivo Tier0 llamar a un programa del iSeries y acceder a los datos que se devuelven una vez ejecutado el programa.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más detalles, consulte la información de requisitos e instalación de ToolboxME para iSeries.

Para utilizar el método ProgramCall.run(), tiene que proporcionar los siguientes parámetros:

- El sistema en el que desea ejecutar el programa.
- El nombre del documento PCML (Program Call Markup Language).
- El nombre del programa que desea ejecutar.
- La tabla hash que contiene el nombre de uno o varios parámetros del programa que desea establecer y los valores asociados.
- La matriz de tipo serie que contiene el nombre de los parámetros que deben devolverse una vez ejecutado el programa.

ProgramCall utiliza PCML para describir los parámetros de entrada y salida del programa. El archivo PCML debe estar en la misma máquina que MEServer y es preciso que haya una entrada para el directorio que contiene el archivo PCML en la CLASSPATH de esa máquina.

Debe registrar cada documento PCML en MEServer. Registrar un documento PCML consiste simplemente en indicar a MEServer qué programa definido en PCML desea ejecutar. Registre el documento PCML en tiempo de ejecución o al iniciar MEServer.

Para obtener más información sobre la tabla hash que contiene los parámetros del programa o cómo registrar un documento PCML, consulte el javadoc de ProgramCall de ToolboxME para iSeries. Para obtener más información sobre PCML, consulte PCML (Program Call Markup Language).

El siguiente ejemplo muestra cómo se utiliza la clase ProgramCall a fin de emplear el dispositivo Tier 0 para ejecutar un programa en un servidor:

```
   // Llame a programas.
   AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");

   String pcmlName = "qsyrusri.pcml"; // El documento PCML que describe el
programa que queremos utilizar.
   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);

       // Obtenga y visualice el perfil de usuario.
       System.out.println("Perfil de usuario: " + valuesToGet[0]);

       // Obtenga y visualice la fecha en un formato legible.
       char[] c = valuesToGet[1].toCharArray();
       System.out.println("Fecha de último inicio de sesión: " +
c[3]+c[4]+"/"+c[5]+c[6]+"/"+c[1]+c[2] );

       // Obtenga y visualice la hora en un formato legible.
       char[] d = valuesToGet[2].toCharArray();
       System.out.println("Hora de último inicio de sesión: " +
d[0]+d[1]+":"+d[2]+d[3]);

       // Obtenga y visualice la información de inicio de sesión.
       System.out.println("Información de inicio de sesión: " +
valuesToGet[3] );
   }
   catch (MEException te)
   {
       // Maneje la excepción.
   }
   catch (IOException ioe)
   {
       // Maneje la excepción.
   }

   // Se ha terminado de trabajar con el objeto del sistema.
   system.disconnect();
```

«

Clases JdbcMe

Las clases de ToolboxME para iSeries proporcionan soporte para JDBC, que incluye el soporte para el paquete java.sql. Las clases están pensadas para utilizarse en un programa que se ejecute en un dispositivo Tier 0.

En los apartados siguientes se trata cómo acceder a los datos y utilizarlos y se describe el contenido de JdbcMe, con enlaces a información acerca de las clases JdbcMe individuales.

Cómo acceder a los datos y utilizarlos

Al utilizar un dispositivo Tier0 para acceder a los datos y actualizarlos, lo que el usuario desea es que funcione exactamente como si estuviera sentado frente a un sistema de la oficina. Sin embargo, gran parte del desarrollo de los dispositivos Tier0 se centra en la sincronización de los datos. Con la sincronización de datos, cada dispositivo Tier0 tiene una copia de datos específicos de la base de datos principal. Periódicamente los usuarios sincronizan los datos de cada dispositivo con la base de datos principal.

La sincronización de datos no funciona correctamente con los datos dinámicos. Para trabajar con datos dinámicos se necesita un rápido acceso a los datos actualizados. Tener que esperar para acceder a los datos sincronizados no es una opción válida para muchas empresas. Además, las exigencias de software y hardware de los servidores y dispositivos para los principales datos síncronos pueden ser importantes.

Para ayudar a resolver los problemas inherentes al modelo de sincronización de datos, las clases JdbcMe de ToolboxME para iSeries permiten llevar a cabo actualizaciones dinámicas y acceder a la base de datos principal, pero siguen permitiendo el almacenamiento de datos fuera de línea. La aplicación puede tener acceso a datos de interés fuera de línea sin sacrificar la posibilidad de que las actualizaciones dinámicas se conviertan de inmediato en parte de la base de datos principal. Este enfoque medio proporciona las ventajas tanto del modelo de datos síncronos como del modelo de datos dinámicos.

Contenido de JdbcMe

Por definición, cualquier tipo de controlador de un dispositivo Tier0 debe ser muy pequeño. Sin embargo, la API JDBC es muy grande. Las clases JdbcMe tenían que ser sumamente pequeñas pero seguir soportando lo suficiente de las interfaces JDBC para que los dispositivos Tier0 pudieran emplearlas para llevar a cabo tareas significativas.

Las clases JdbcMe ofrecen las siguientes funciones JDBC:

- Posibilidad de insertar o actualizar datos
- Control de transacciones y posibilidad de modificar los niveles de aislamiento de transacción
- Conjuntos de resultados que son desplazables y actualizables
- Soporte SQL para las llamadas a procedimientos almacenados y desencadenantes de unidad

Además, las clases JdbcMe tienen algunas características exclusivas:

- Un controlador universal que hace posible la consolidación en un solo punto del lado del servidor de la mayoría de los detalles de configuración.
- Un mecanismo estándar para el almacenamiento persistente de los datos fuera de línea.

JdbcMe incluye las clases siguientes:

- JdbcMeConnection
- JdbcMeDriver
- JdbcMeException
- JdbcMeLiveResultSet
- JdbcMeOfflineData

- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData
- JdbcMeStatement

Conformidad con SQL

ToolboxME para iSeries proporciona un paquete java.sql que cumple la especificación JDBC pero sólo contiene el conjunto mínimo de clases y métodos de utilidad. El hecho de proporcionar un conjunto mínimo de funciones sql permite que las clases JdbcMe tengan un tamaño pequeño pero que, al mismo tiempo, puedan ser suficientemente útiles para llevar a cabo las tareas JDBC comunes.

»Utilización de ToolboxME para iSeries para conectarse a una base de datos del servidor de sistema principal

La clase JdbcMeConnection proporciona un subconjunto de las funciones disponibles en la clase AS400JDBCConnection de Toolbox para Java. Utilcie JdbcMeConnection para permitir al dispositivo Tier0 acceder a las bases de datos DB2 Universal Database (UDB) del servidor de sistema principal.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más detalles, consulte la información de requisitos e instalación de ToolboxME para iSeries.

Utilice JdbcMeDriver.getConnection() para conectarse a la base de datos del servidor. El método getConnection() toma como argumento una serie de URL (localizador uniforme de recursos), el ID de usuario y la contraseña. El gestor de controladores JDBC del servidor de sistema principal intenta localizar un controlador que pueda conectarse a la base de datos representada por el URL. JdbcMeDriver emplea la siguiente sintaxis para el URL:

```
jdbc:as400://nombre-servidor/esquema-por-omisión;meserver=<servidor>[:puerto];[otras
propiedades];
```

Debe especificar un nombre de servidor; de lo contrario, JdbcMeDriver lanza una excepción. El esquema por omisión es opcional. Si no especifica ningún puerto, JdbcMeDriver utiliza el puerto 3470. Además, puede establecer diversas propiedades de JDBC en el URL. Para establecer propiedades, emplee la sintaxis siguiente:

```
      nombre1=valor1;nombre2=valor2;...
```

En Propiedades de JDBC encontrará una lista completa de las propiedades soportadas por JdbcMeDriver.

Ejemplos: utilización de JdbcMeDriver para conectarse a un servidor

Ejemplo 1: conectarse a la base de datos del servidor sin especificar ningún esquema por omisión, puerto ni propiedad de JDBC. El ID de usuario y la contraseña se especifican como parámetros en el método.

```
         // Conéctese al sistema 'mysystem'. No se ha especificado
         // ningún esquema, puerto ni propiedad.
    Connection c  = JdbcMeDriver.getConnection
         ("jdbc:as400://mysystem.helloworld.com;
           meserver=myMeServer;"
                       "auser",
                       "apassword");
```

Ejemplo 2: conectarse a la base de datos del servidor especificando el esquema y las propiedades de JDBC. El ID de usuario y la contraseña se especifican como parámetros en el método.

```
         // Conéctese al sistema 'mysystem'. Especifique un esquema
         // y dos propiedades de JDBC. No especifique ningún puerto.
    Connection c2 = JdbcMeDriver.getConnection
         ("jdbc:as400://mysystem.helloworld.com/mySchema;
          meserver=myMeServer;
          naming=system;
          errors=full;"
                       "auser",
                       "apassword");
```

Ejemplo 3: conectarse a la base de datos del servidor; las propiedades (entre ellas el ID de usuario y la contraseña) se especifican mediante un URL (localizador uniforme de recursos).

```
         // Conéctese utiliziando propiedades. Las propiedades se establecen
en el URL
         // en lugar de mediante un objeto de propiedades.
     Connection c = DriverManager.getConnection
```

```
("jdbc:as400://mySystem;
  meserver=myMeServer;
  naming=sql;
  errors=full;
  user=auser;
  password=apassword");
```

Ejemplo 4: desconectarse de la base de datos. Utilice el método close() en el objeto de conexión para desconectarse del servidor:

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

Clase JdbcMeDriver

La clase JdbcMeDriver proporciona un subconjunto de las funciones disponibles en la clase AS400JDBCDriver de Toolbox para Java. Utilice JdbcMeDriver en la aplicación cliente Tier0 para ejecutar sentencias SQL sencillas que no tengan parámetros y obtener los objetos ResultSet que generen las sentencias.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más información, consulte Bajar y configurar ToolboxME para iSeries.

JdbcMeDriver no se registra explícitamente; en su lugar, la propiedad **driver** que especifique en el URL del método JdbcMeConnection.getConnection() determinará el controlador. Por ejemplo, para cargar el controlador JDBC de IBM Developer Kit para Java (denominado controlador "nativo"), utilice el código siguiente:

```
Connection c = JdbcMeDriver.getConnection
     ("jdbc:as400://mysystem.helloworld.com;
       meserver=myMeServer;
       driver=native;
       user=auser;
       password=apassword");
```

El controlador JDBC de IBM Toolbox para Java no requiere un objeto AS400 como parámetro de entrada, como lo requieren las demás clases de IBM Toolbox para Java que obtienen datos de un servidor. Sin embargo, se utiliza un objeto AS400 internamente y debe proporcionar explícitamente un ID de usuario y una contraseña. Especifique el ID de usuario y la contraseña en el URL o mediante los parámetros del método getConnection().

Si desea ver ejemplos de cómo se utiliza getConnection(), consulte JDBCMeConnection.

Conjuntos de resultados

Las clases de conjuntos de resultados de ToolboxME para iSeries son:

- JdbcMeLiveResultSet
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData

JdbcMeLiveResultSet y JdbcMeOfflineResultSet contienen las mismas funciones, con la excepción de que:

- JdbcMeLiveResultSet recupera datos efectuando una llamada a la base de datos del servidor.
- JdbcMeOfflineResultSet recupera datos de la base de datos del dispositivo local.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más información, consulte Bajar y configurar ToolboxME para iSeries.

JdbcMeLiveResultSet

La clase JdbcMeLiveResultSet proporciona un subconjunto de las funciones disponibles en la clase AS400JDBCResultSet de Toolbox para Java. Utilice JdbcMeLiveResultSet en la aplicación cliente Tier0 para acceder a una tabla de datos generada ejecutando una consulta.

JdbcMeLiveResultSet recupera las filas de la tabla por orden. Dentro de una fila, puede acceder a los valores de columna en cualquier orden. JdbcMeLiveResultSet contiene métodos que permiten llevar a cabo las acciones siguientes:

- Recuperar datos de diversos tipos que están almacenados en el conjunto de resultados
- Mover el cursor a la fila que especifique (la fila anterior, la fila actual, la fila siguiente, etc.)
- Insertar, actualizar y suprimir filas
- Actualizar columnas (utilizando valores String e int)
- Recuperar el objeto ResultSetMetaData que describe las columnas del conjunto de resultados

Un cursor, que es un puntero interno utilizado por un conjunto de resultados, señala a la fila del conjunto de resultados a la que está accediendo el programa Java. JDBC 2.0 proporciona métodos adicionales para acceder a posiciones específicas dentro de una base de datos:

Posiciones de cursor desplazables	
absolute	moveToInsertRow
first	previous
last	relative
moveToCurrentRow	

Posibilidades de desplazamiento

Si un conjunto de resultados se crea mediante la ejecución de una sentencia, es posible moverse (desplazarse) por las filas de una tabla en sentido hacia atrás (de la última a la primera) o hacia delante (de la primera a la última).

Un conjunto de resultados que dé soporte a este movimiento se llama desplazable. Los conjuntos de resultados desplazables admiten también el posicionamiento relativo y el absoluto. El posicionamiento relativo le permite moverse a una fila del conjunto de resultados especificando una posición relativa a la fila actual. El posicionamiento absoluto le permite moverse directamente a una fila especificando la posición que dicha fila tiene en el conjunto de

resultados.

Con JDBC 2.0, se dispone de dos posibilidades de desplazamiento adicionales al trabajar con la clase ResultSet: conjuntos de resultados no sensibles al desplazamiento y sensibles al desplazamiento.

Un conjunto de resultados no sensible al desplazamiento suele no ser sensible a los cambios realizados mientras está abierto, mientras que el conjunto de resultados sensible al desplazamiento es sensible a los cambios.»El controlador JDBC de IBM Toolbox para Java no soporta conjuntos de resultados no sensibles al desplazamiento«

Conjuntos de resultados actualizables

En la aplicación, puede utilizar conjuntos de resultados que emplean ya sea una concurrencia de solo lectura (no pueden realizarse actualizaciones en los datos) o una concurrencia actualizable (permite realizar actualizaciones en los datos y puede utilizar bloqueos de escritura de base de datos para controlar las distintas transacciones que acceden a un mismo elemento de datos). En un conjunto de resultados actualizable, las filas se pueden actualizar, insertar y suprimir.

En Resumen de métodos encontrará una lista completa de los métodos de actualización que están disponibles en JdbcMeResultSet.

Ejemplo: conjuntos de resultados actualizables

El ejemplo que figura a continuación muestra cómo se utiliza un conjunto de resultados que permite realizar actualizaciones en los datos (concurrencia de actualización) y realizar cambios en el conjunto de resultados mientras permanece abierto (sensible al desplazamiento).

```
          // Conéctese al servidor.
     Connection c = JdbcMeDriver.getConnection
          ("jdbc:as400://mySystem;
            meserver=myMeServer;
            user=auser;
            password=apassword");

         // Cree un objeto Statement. Establezca la concurrencia
         // del conjunto de resultados en actualizable.
     Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                ResultSet.CONCUR_UPDATABLE);

         // Ejecute una consulta. El resultado se coloca
         // en un objeto ResultSet.
     ResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE
FOR UPDATE");

         // Itere por las filas de ResultSet. A medida que
         // lea la fila, se actualizará con un nuevo ID.
     int newId = 0;
     while (rs.next ())
     {

              // Obtenga los valores de ResultSet. El primer valor
              // es una serie y el segundo valor es un entero.
         String name = rs.getString("NAME");
         int id = rs.getInt("ID");

         System.out.println("Nombre = " + name);
         System.out.println("ID antiguo = " + id);

         // Actualice el ID con un nuevo entero.
```

```
        rs.updateInt("ID", ++newId);

        // Envíe las actualizaciones al servidor.
        rs.updateRow ();

        System.out.println("ID nuevo = " + newId);
    }

     // Cierre la sentencia y la conexión.
    s.close();
    c.close();
```

Clase JdbcMeOfflineResultSet

La clase JdbcMeOfflineResultSet proporciona un subconjunto de las funciones disponibles en la clase AS400JDBCResultSet de Toolbox para Java. Utilice JdbcMeOfflineResultSet en la aplicación cliente Tier0 para acceder a una tabla de datos generada ejecutando una consulta.

Utilice la clase JdbcMeOfflineResultSet para trabajar con datos que residen en el dispositivo Tier0. Los datos que residen en el dispositivo ya podían residir ahí o puede haberlos colocado en esa ubicación llamando al método JdbcMeStatement.executeToOfflineData(). El método executeToOfflineData() baja y almacena en el dispositivo todos los datos que satisfacen la consulta. A continuación puede utilizar la clase JdbcMeOfflineResultSet para acceder a los datos almacenados.

JdbcMeOfflineResultSet contiene métodos que permiten llevar a cabo las acciones siguientes:

- Recuperar datos de diversos tipos que están almacenados en el conjunto de resultados
- Mover el cursor a la fila que especifique (la fila anterior, la fila actual, la fila siguiente, etc.)
- Insertar, actualizar y suprimir filas
- Actualizar columnas (utilizando valores String e int)
- Recuperar el objeto ResultSetMetaData que describe las columnas del conjunto de resultados

Puede ofrecer la posibilidad de sincronizar la base de datos del dispositivo local con la base de datos del servidor iSeries utilizando las funciones existentes en las clases JdbcMe.

Clase JdbcMeResultSetMetaData

La clase JdbcMeResultSetMetaData proporciona un subconjunto de las funciones disponibles en la clase AS400JDBCResultSetMetaData de Toolbox para Java. Utilice JdbcMeResultSetMetaData en la aplicación cliente Tier0 para determinar los tipos y las propiedades de las columnas de un objeto JDBCResultSet.

El siguiente ejemplo muestra cómo se utiliza la clase JdbcMeResultSetMetaData:

```
     // Conéctese al servidor.
    Connection c = JdbcMeDriver.getConnection
     ("jdbc:as400://mySystem;
       meserver=myMeServer;
       user=auser;
       password=apassword");

    // Cree un objeto Statement.
    Statement s = c.createStatement();

    // Ejecute una consulta. El resultado se coloca en un objeto
ResultSet.
```

```
     JdbcMeLiveResultSet rs = s.executeQuery ("SELECT NAME,ID FROM
MYLIBRARY.MYTABLE");

         // Itere por las filas de ResultSet.
     while (rs.next ())
     {

          // Obtenga los valores de ResultSet. El primer valor es
          // una serie y el segundo valor es un entero.
         String name = rs.getString("NAME");
         int id = rs.getInt("ID");

         System.out.println("Nombre = " + name);
         System.out.println("ID = " + id);
     }

      // Cierre la sentencia y la conexión.
     s.close();
     c.close();
```

«

Clase JdbcMeOfflineData

La clase JdbcMeOfflineData es un depósito de datos fuera de línea pensado para utilizarse en un dispositivo Tier0. El depósito es genérico, con independencia del perfil y la máquina virtual Java que se utilice. Para obtener más información, consulte los conceptos de ToolboxME para iSeries.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más información, consulte Bajar y configurar ToolboxME para iSeries.

La clase JdbcMeOfflineData proporciona métodos que permiten llevar a cabo las acciones siguientes:

- Crear un depósito de datos fuera de línea
- Abrir un depósito existente
- Obtener el número de registros del depósito
- Obtener y suprimir registros individuales
- Actualizar registros
- Añadir un registro al final del depósito
- Cerrar el depósito

Para ver un ejemplo de cómo se utiliza la clase JdbcMeOfflineData, consulte el ejemplo de ToolboxME para iSeries: cómo se utiliza ToolboxME para iSeries, MIDP e IBM Toolbox para Java.

»Clase JdbcMeStatement

La clase JdbcMeStatement proporciona un subconjunto de las funciones disponibles en la clase AS400JDBCStatement de Toolbox para Java. Utilice JdbcMeStatement en la aplicación cliente Tier0 para ejecutar sentencias SQL sencillas que no tengan parámetros y obtener los objetos ResultSet que generen las sentencias.

Nota: para utilizar las clases de ToolboxMe para iSeries, debe bajar y configurar por separado el componente ToolboxME para iSeries. Para obtener más información, consulte Bajar y configurar ToolboxME para iSeries.

Clase Statement

Utilice JdbcMeConnection.createStatement() para crear nuevos objetos Statement.

El ejemplo que sigue muestra cómo se utiliza un objeto JdbcMeStatement:

```
          // Conéctese al servidor.
     JdbcMeConnection c = JdbcMeDriver.getConnection(
          "jdbc:as400://mysystem.helloworld.com/mylibrary;
           naming=system;
           errors=full;
           meserver=myMeServer;
           user=auser;
           password=apassword");

         // Cree un objeto Statement.
     JdbcMeStatement s = c.createStatement();

          // Ejecute una sentencia SQL que cree una tabla en la base de
datos.
     s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID
INTEGER)");

          // Ejecute una sentencia SQL que inserte un registro en la tabla.
     s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES
('DAVE', 123)");

          // Ejecute una sentencia SQL que inserte un registro en la tabla.
     s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES
('CINDY', 456)");

         // Ejecute una consulta SQL en la tabla.
     JdbcMeLiveResultSet rs = s.executeQuery("SELECT * FROM
MYLIBRARY.MYTABLE");

          // Cierre la sentencia y la conexión.
     s.close();
     c.close();
```

«

»Crear y ejecutar un programa de ToolboxME para iSeries

Esta información le permitirá editar, compilar y ejecutar el programa de ToolboxME para iSeries de ejemplo. También puede utilizar esta información a modo de guía general para crear, probar y ejecutar los ejemplos de trabajo de ToolboxME para iSeries y sus propias aplicaciones de ToolboxME para iSeries.

El programa de ejemplo utiliza la máquina virtual K (KVM) y permite al usuario llevar a cabo cualquier consulta JDBC. El usuario puede llevar a cabo acciones JDBC (next, previous, close, commit y rollback) para el resultado de la consulta.

Antes de empezar a crear cualquiera de los ejemplos de ToolboxME para iSeries, asegúrese de que el entorno cumple los requisitos de ToolboxME para iSeries.

Crear el ejemplo de ToolboxME para iSeries

Para crear el programa de ejemplo de ToolboxME para iSeries para el dispositivo Tier0, siga estos pasos:

1. Copie el código Java para el ejemplo de ToolboxME para iSeries, denominado JdbcDemo.java.
2. En el editor de Java o texto elegido, cambie las partes del código como se indica en los comentarios del programa y guarde el archivo con el nombre JdbcDemo.java.

   Nota: plantéese la posibilidad de utilizar una herramienta de desarrollo de aplicaciones inalámbricas, con lo que llevar a cabo los pasos restantes es más sencillo. Algunas herramientas de desarrollo de aplicaciones inalámbricas pueden compilar, preverificar y construir el programa en un solo paso y, a continuación, ejecutarlo automáticamente en un emulador.
3. Compile JdbcDemo.java y asegúrese de que apunta al archivo .jar que contiene las clases de KVM.
4. Preverifique el archivo ejecutable mediante la herramienta de desarrollo de aplicaciones inalámbricas o el mandato de preverificar de Java.
5. Construya el tipo de archivo ejecutable adecuado para el sistema operativo del dispositivo Tier0. Por ejemplo, para el sistema operativo Palm, construiría un archivo denominado JdbcDemo.prc.
6. Pruebe el programa. Si ha instalado un emulador, puede probar el programa y ver cómo se visualizará ejecutándolo en el emulador.

   Nota: si prueba el programa en el dispositivo inalámbrico y no utiliza una herramienta de desarrollo de aplicaciones inalámbricas, asegúrese de precargar la máquina virtual Java elegida o el MIDP en el dispositivo.

Consulte los conceptos acerca de ToolboxME para iSeries para obtener información relacionada sobre conceptos, herramientas de desarrollo de aplicaciones inalámbricas y emuladores.

Ejecutar el ejemplo de ToolboxME para iSeries

Para ejecutar el programa de ejemplo de ToolboxME para iSeries en el dispositivo Tier0, efectúe estas tareas:

- Cargue el archivo ejecutable para el dispositivo siguiendo las instrucciones facilitadas por el fabricante del dispositivo Tier0.
- Inicie MEServer
- Ejecute el programa JdbcDemo en el dispositivo Tier0 pulsando el icono JdbcDemo.«

Ejemplos de trabajo de ToolboxME para iSeries

Los siguientes ejemplos de trabajo de ToolboxMe para iSeries muestran modos de utilizar ToolboxME para iSeries con MIDP (Mobile Information Device Profile). Utilice los enlaces siguientes para ver los archivos fuente de ejemplo seleccionados o bajar todos los archivos fuente necesarios para crear las aplicaciones inalámbricas de ejemplo de trabajo:

Ejemplo: ToolboxME para iSeries, MIDP y JDBC

Ejemplo: ToolboxME para iSeries, MIDP e IBM Toolbox para Java

Bajar los ejemplos de trabajo de ToolboxME para iSeries

Para obtener más información sobre cómo crear una aplicación de ToolboxME para iSeries, consulte Crear y ejecutar un programa de ToolboxME para iSeries.

Preguntas habituales (FAQ)

Las preguntas habituales (FAQ) de IBM Toolbox para Java dan respuesta a cuestiones relacionadas con la optimización del rendimiento de IBM Toolbox para Java, la resolución de problemas, la utilización de JDBC y otros temas:

- Preguntas habituales de IBM Toolbox para Java  : encuentre la respuesta a muchos tipos de preguntas, entre ellas cómo aumentar el rendimiento, utilizar el OS/400, solucionar los problemas, etc.
- Preguntas habituales de JDBC de IBM Toolbox para Java  : encuentre la respuesta a preguntas acerca de cómo utilizar JDBC con Toolbox para Java.

Ejemplos de código

En la lista siguiente se proporcionan enlaces con los puntos de entrada de muchos de los ejemplos empleados en el tema de IBM Toolbox para Java.

Clases de acceso	Beans	Caja de Herramientas Gráfica
Clases HTML	PCML	Clases ReportWriter
Clases de recursos	RFML	Clases de seguridad
Clases de servlets	Ejemplos simples	Consejos para la programación
ToolboxMe para iSeries	Clases de utilidades	Clases de vaccess

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplos: clases de acceso

En esta sección figura una lista de los ejemplos de código que se proporcionan en toda la documentación de las clases de acceso.

AS400JPing

- Ejemplo: cómo se utiliza AS400JPing en un programa Java

BidiTransform

- Ejemplo: cómo se utiliza la clase AS400BidiTransform para transformar texto bidireccional

CommandCall

- Ejemplo: cómo se utiliza CommandCall para ejecutar un mandato en el servidor
- Ejemplo: cómo se utiliza CommandCall para solicitar el nombre del servidor y el mandato que se ha de ejecutar e imprimir el resultado

ConnectionPool

- Ejemplo: cómo se utiliza AS400ConnectionPool para crear conexiones con el servidor

DataArea

- Ejemplo: crear y escribir en un área de datos decimales

Conversión y descripción de datos

- Ejemplo: cómo se utilizan las clases FieldDescription, RecordFormat y Record
- Ejemplo: poner datos en una cola
- Ejemplo: leer datos de una cola

DataQueue

- Ejemplo: crear un objeto DataQueue, leer datos y desconectar
- Ejemplo: poner datos en una cola
- Ejemplo: leer datos de una cola

Certificado digital

- Ejemplo: listar los certificados digitales pertenecientes a un usuario

EnvironmentVariable

- Ejemplo: crear, establecer y obtener variables de entorno

Excepciones

- Ejemplo: capturar una excepción lanzada, recuperar el código de retorno y visualizar el texto de la excepción

FTP

- Ejemplo: cómo se utiliza la clase FTP para copiar un conjunto de archivos de un directorio del servidor
- Ejemplo: cómo se utiliza la clase AS400FTP para copiar un conjunto de archivos de un directorio

Sistema de archivos integrado

- Ejemplos: cómo se utiliza IFSFile
- Ejemplo: cómo se utiliza el método IFSFile.listFiles() para listar el contenido de un directorio
- Ejemplo: cómo se utilizan las clases IFSFile para copiar archivos
- Ejemplo: cómo se utilizan las clases IFSFile para listar el contenido de un directorio
- Ejemplo: cómo se utiliza IFSJavaFile, en lugar de java.io.File
- Ejemplo: cómo se utilizan las clases IFSFile para listar el contenido de un directorio en el servidor

JavaApplicationCall

- Ejemplo: ejecutar en el servidor un programa desde el cliente cuya salida es "¡Hola a todos!"

JDBC

- Ejemplo: cómo se utiliza el controlador JDBC para crear y llenar con datos una tabla
- Ejemplo: cómo se utiliza el controlador JDBC para consultar una tabla y enviar su contenido a la salida

Trabajos

- Ejemplo: recuperar y cambiar la información de trabajo utilizando la antememoria
- Ejemplo: listar todos los trabajos activos
- Ejemplo: imprimir todos los mensajes de las anotaciones de trabajo correspondientes a un usuario específico
- Ejemplo: listar la información de identificación de trabajo correspondiente a un usuario específico
- Ejemplo: obtener una lista de los trabajos existentes en el servidor y listar el estado del trabajo y el identificador del mismo
- Ejemplo: visualizar los mensajes de las anotaciones de trabajo correspondientes a un trabajo perteneciente al usuario actual

Cola de mensajes

- Ejemplo: cómo se utiliza el objeto cola de mensajes
- Ejemplo: imprimir el contenido de la cola de mensajes
- Ejemplo: cómo recuperar e imprimir un mensaje
- Ejemplo: listar el contenido de la cola de mensajes
- Ejemplo: cómo se utiliza AS400Message con CommandCall
- Ejemplo: cómo se utiliza AS400Message con ProgramCall

NetServer

- Ejemplo: cómo se utiliza un objeto NetServer para cambiar el nombre del NetServer

Imprimir

- Ejemplo: crear un archivo en spool a partir de una corriente de entrada
- Ejemplo: generar una corriente de datos SCS utilizando la clase SCS3812Writer
- Ejemplo: leer un archivo en spool existente
- Ejemplo: listar asíncronamente todos los archivos en spool utilizando la interfaz PrintObjectListListener
- Ejemplo: listar asíncronamente todos los archivos en spool *sin* utilizar la interfaz PrintObjectListListener
- Ejemplo: listar síncronamente todos los archivos en spool

Permiso

- Ejemplo: establecer la autorización de un objeto de AS400

Llamada a programa

- Ejemplo: cómo se utiliza ProgramCall
- Ejemplo: cómo se utiliza ProgramCall para recuperar el estado del sistema
- Ejemplo: pasar datos de parámetro con un objeto Programparameter

QSYSObjectPathName

- Ejemplo: construir un nombre de sistema de archivos integrado
- Ejemplo: cómo se utiliza QSYSObjectPathName.toPath() para construir un nombre de objeto AS400
- Ejemplo: cómo se utiliza QSYSObjectPathName para analizar el nombre de vía de acceso del sistema de archivos integrado

Acceso a nivel de registro

- Ejemplo: acceder secuencialmente a un archivo
- Ejemplo: cómo se utilizan las clases de acceso a nivel de registro para leer un archivo

- Ejemplo: cómo se utilizan las clases de acceso a nivel de registro para leer registros por clave
- Ejemplo: cómo se utiliza la clase LineDataRecordWriter

Llamada a programa de servicio

- Ejemplo: cómo se utiliza ServiceProgramCall para llamar a un procedimiento

Estado del sistema

- Ejemplo: cómo se utiliza la puesta en antememoria con la clase SystemStatus

Agrupación del sistema

- Ejemplo: establecer el tamaño máximo de faltas para una agrupación del sistema

SystemValue

- Ejemplo: cómo se utiliza SystemValue y SystemValueList

Rastreo

- Ejemplo: cómo se utiliza el método Trace.setTraceOn()
- Ejemplo: procedimiento preferido de uso del rastreo
- Ejemplo: cómo se utiliza el rastreo de componentes

Grupos de usuarios

- Ejemplo: recuperar una lista de usuarios
- Ejemplo: listar todos los usuarios de un grupo

Espacio de usuario

- Ejemplo: cómo se crea un espacio de usuario

Ejemplo: cómo se utiliza CommandCall

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo de llamada a mandato. Este programa solicita al usuario
// el nombre del servidor y el mandato que se debe ejecutar y, a
// continuación, imprime el resultado del mandato.
//
// Este fuente es un ejemplo de "CommandCall" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class CommandCallExample extends Object
{
   public static void main(String[] parmeters)
   {

    // Se ha creado un lector para obtener entrada del usuario
      BufferedReader inputStream = new BufferedReader(new
InputStreamReader(System.in),1);



      // Declare variables para nombre de sistema y mandato que se ejecutará
      String systemString  = null;
      String commandString = null;

      System.out.println( " " );



      // Obtenga el nombre del sistema y el mandato que se ejecutará del
usuario
                           try
      {
        System.out.print("Nombre del sistema: ");
         systemString = inputStream.readLine();

         System.out.print("Mandato: ");
         commandString = inputStream.readLine();
      }
            catch (Exception e) {};

      System.out.println( " " );



      // Cree un objeto AS400. Es el sistema al que se envía el mandato
      AS400 as400 = new AS400(systemString);
```

```
      // Cree un objeto de llamada a mandato especificando el servidor que
      // recibirá el mandato.
      CommandCall command = new CommandCall( as400 );



                           try
      {
         // Ejecute el mandato.
         if (command.run(commandString))
             System.out.print( "Mandato satisfactorio" );
          else
             System.out.print( "Mandato anómalo" );



         // Si se generan mensajes a partir del mandato, imprímalos
         AS400Message[] messagelist = command.getMessageList();

         if (messagelist.length > 0)
         {
             System.out.println( ", mensajes del mandato:" );
             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( "El mandato " + command.getCommand() + " no se
ha ejecutado" );
      }

       System.exit(0);
   }
}
```

Ejemplo: cómo se utiliza AS400ConnectionPool

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de AS400ConnectionPooling. Este programa utiliza una
AS400ConnectionPool
// para crear conexiones con un sistema iSeries.
// Sintaxis de mandato:
//    AS400ConnectionPooling sistema MiIDUsuario MiContraseña
//
// Por ejemplo:
//    AS400ConnectionPooling MySystem MyUserId MyPassword
//
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class AS400ConnectionPooling
{
   public static void main (String[] parameters)
   {
        // Compruebe los parámetros de entrada.
      if (parameters.length != 3)
      {
            System.out.println("");
         System.out.println("Utilización:");
            System.out.println("");
         System.out.println("   AS400ConnectionPooling sistema IDUsuario
contraseña");
            System.out.println("");
            System.out.println("");
            System.out.println("Por ejemplo:");
            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
      {
                        // Cree una AS400ConnectionPool.
         AS400ConnectionPool testPool = new AS400ConnectionPool();

         // Establezca un máximo de 128 conexiones para esta agrupación.
         testPool.setMaxConnections(128);

         // Establezca un tiempo máximo de vida de 30 minutos para las
conexiones.
         testPool.setMaxLifetime(1000*60*30);     // 30 min tiempo máximo de
vida desde que se creen.
```

```
         // Preconectar 5 conexiones con el servicio AS400.COMMAND.
         testPool.fill(system, userId, password, AS400.COMMAND, 1);
    System.out.println();
         System.out.println("1 conexión preconectada con el servicio
AS400.COMMAND");

         // Llame a getActiveConnectionCount y getAvailableConnectionCount
para ver cuántas
         // conexiones están en uso y disponibles para un sistema concreto.
         System.out.println("Número de conexiones activas: " +
testPool.getActiveConnectionCount(system, userId));
         System.out.println("Número de conexiones disponibles para el uso: "
+ testPool.getAvailableConnectionCount(system, userId));

                        // Cree una conexión con el servicio AS400.COMMAND.
(Emplee las constantes de número de servicio
                        // definidas en la clase AS400 (FILE, PRINT, COMMAND,
DATAQUEUE, etc.))
         // Dado que ya se han llenado las conexiones, el tiempo que suele
dedicarse a conectar con el
         // servicio de mandatos se evita.
         AS400 newConn1 = testPool.getConnection(system, userId, password,
AS400.COMMAND);

    System.out.println();
         System.out.println("getConnection proporciona una conexión
existente al usuario");
         System.out.println("Número de conexiones activas: " +
testPool.getActiveConnectionCount(system, userId));
         System.out.println("Número de conexiones disponibles para el uso: "
+ testPool.getAvailableConnectionCount(system, userId));

         // Cree un nuevo objeto de llamada a mandato y ejecute un mandato.
         CommandCall cmd1 = new CommandCall(newConn1);
         cmd1.run("CRTLIB FRED");

         // Devuelva la conexión a la agrupación.
         testPool.returnConnectionToPool(newConn1);

    System.out.println();
         System.out.println("Se ha devuelto una conexión a la agrupación");
         System.out.println("Número de conexiones activas: " +
testPool.getActiveConnectionCount(system, userId));
         System.out.println("Número de conexiones disponibles para
reutilizar: " + testPool.getAvailableConnectionCount(system, userId));

         // Cree una conexión con el servicio AS400.COMMAND. Esto devolverá
el mismo
         // objeto que se ha indicado anteriormente para volverse a
utilizar.
         AS400 newConn2 = testPool.getConnection(system, userId, password,
AS400.COMMAND);

    System.out.println();
         System.out.println("getConnection proporciona una conexión
existente al usuario");
         System.out.println("Número de conexiones activas: " +
testPool.getActiveConnectionCount(system, userId));
         System.out.println("Número de conexiones disponibles para
reutilizar: " + testPool.getAvailableConnectionCount(system, userId));

         // Cree una conexión con el servicio AS400.COMMAND. Esto creará una
```

```
nueva
         // conexión ya que no hay ninguna conexión en la agrupación para
reutilizar.
         AS400 newConn3 = testPool.getConnection(system, userId, password,
AS400.COMMAND);

    System.out.println();
         System.out.println("getConnection crea una nueva conexión ya que no
hay conexiones disponibles");
         System.out.println("Número de conexiones activas: " +
testPool.getActiveConnectionCount(system, userId));
         System.out.println("Número de conexiones disponibles para
reutilizar: " + testPool.getAvailableConnectionCount(system, userId));

         // Cierre la agrupación de prueba.
         testPool.close();
      }
        catch (Exception e)
      {
         // Si alguna de las operaciones anteriores ha fallado, indique que
las operaciones de la agrupación han fallado
         // y envíe a la salida la excepción.

         System.out.println("Las operaciones de la agrupación han fallado");
        System.out.println(e);
      e.printStackTrace();
      }
   }
}
```

Ejemplo: cómo se utilizan las clases DataQueue para poner datos en una cola

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo de cola de datos. Este programa utiliza la clase DataQueue para
// poner registros en una cola de datos.
//
// Este ejemplo utiliza las clases de registro y de formato de registro para
// poner datos en la cola. Los datos de serie se convierten de Unicode a
ebcdic
// y los números se convierten de formato Java a formato del servidor. Dado
que
// los datos se convierten, las entradas de cola de datos pueden ser leídas
por un
// programa del servidor o un programa iSeries Access para Windows y otro
programa Java.
//
// Este es el lado del productor del ejemplo de productor/consumidor. Coloca
// elementos de trabajo en la cola para que el consumidor los procese.
//
// Sintaxis de mandato:
//    DQProducerExample sistema
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQProducerExample extends Object
{
   // Cree un lector para obtener entrada del usuario.
   static BufferedReader inputStream = new BufferedReader(new
InputStreamReader(System.in),1);

  public static void main(String[] parameters)
   {
      System.out.println( " " );

      // De no haberse especificado el sistema, visualizar texto de ayuda y
salir.
      if (parameters.length >= 1)
      {
                           try
         {
         // El primer parámetro es el sistema que contiene la cola de datos.
      String system           = parameters[0];

             // Cree un objeto AS400 para el servidor que tiene la cola de
datos.
                AS400 as400 = new AS400(system);

             // Construya un formato de registro para el formato de la
entrada de la cola de datos.
             // Este formato coincide con el formato de la clase DQConsumer.
```

```
Un
            // registro está formado por los elementos siguientes:
            //    - un número de cuatro bytes -- el número de cliente
            //    - un número de cuatro bytes -- el número de pieza
            //    - una serie de 20 caracteres -- la descripción de la
pieza
            //    - un número de cuatro bytes -- el número de piezas de
este pedido
            // Primero cree los tipos de datos base.
            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");

            // Construya un formato de registro y llénelo con los tipos de
datos base.
            RecordFormat dataFormat = new RecordFormat();
            dataFormat.addFieldDescription(customerNumber);
            dataFormat.addFieldDescription(partNumber);
            dataFormat.addFieldDescription(partName);
            dataFormat.addFieldDescription(quantity);

            // Cree la biblioteca que contiene la cola de datos con
CommandCall.
            CommandCall crtlib = new CommandCall(as400);
            crtlib.run("CRTLIB JAVADEMO");

            // Cree el objeto de cola de datos.
            DataQueue dq = new DataQueue(as400,
"/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

            // Cree la cola de datos por si esta es la primera vez que se
ejecuta
            // el programa. La cola ya existe; la excepción se captura y se
            // omite.
                        try
            {
               dq.create(96);
            }
            catch (Exception e) {};

            // Obtenga el primer campo de datos del usuario.
               System.out.print("Especifique el número de cliente (o 0 para
salir): ");
            int customer = getInt();

            // Mientras haya datos para colocar en la cola.
            while (customer > 0)
            {
               // Obtenga los demás datos de este pedido del usuario.
               System.out.print("Especifique el número de pieza: ");
               int part = getInt();
```

```
                System.out.print("Especifique la cantidad: ");
                int quantityToOrder = getInt();

                String description = "pieza " + part;

                // Cree un registro basado en el formato de registro. El
registro
                // ahora está vacío pero finalmente contendrá los datos.
                Record data = new Record(dataFormat);

                // Establezca los valores recibidos del usuario en el
registro.
                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);

                // Convierta el registro en una matriz de bytes. La matriz
de bytes es
                // lo que se coloca realmente en la cola de datos.
                byte [] byteData = data.getContents();

System.out.println("");                System.out.println("Escribiendo el
registro en el servidor...");
System.out.println("");
                // Escriba el registro en la cola de datos.
                dq.write(byteData);

                // Obtenga el valor siguiente del usuario.
                System.out.print("Especifique el número de cliente (o 0 para
salir): ");
                customer = getInt();
             }
         }
        catch (Exception e)
         {
            // Si alguna de las operaciones anteriores ha fallado, indique
que la
            // operación de la cola de datos ha fallado y envíe a la salida
la excepción.

            System.out.println("La operación de cola de datos ha fallado");
        System.out.println(e);
         }
      }

        // Visualice texto de ayuda si los parámetros son incorrectos.
          else
      {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println(" DQProducter sistema");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println(" sistema = Servidor que
tiene la cola de datos");
System.out.println("");            System.out.println("Por ejemplo:");
System.out.println("");         System.out.println(" DQProducerExample
mySystem");
System.out.println("");System.out.println("");      }
```

```
        System.exit(0);
    }

    // Esta es la subrutina que obtiene una serie de caracteres del usuario
    // y la convierte en un objeto 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("Especifique un número ==>");
          }
       }

       return i;
    }
}
```

Ejemplo: cómo se utilizan las clases DataQueue para leer entradas de una cola de datos

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de cola de datos. Este programa utiliza las clases de cola de
datos para leer
// entradas de una cola de datos en el servidor. Las entradas se colocaron
en la
// cola con el programa de ejemplo DQProducer.
//
// Este es el lado del consumidor del ejemplo de productor/consumidor. Lee
// entradas de la cola y las procesa.
//
// Sintaxis de mandato:
//    DQConsumerExample sistema
//
////////////////////////////////////////////////////////////////////////////


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( " " );

      // De no haberse especificado un sistema, visualizar texto de ayuda y
salir.
      if (parameters.length >= 1)
      {
                           try
         {

         // El primer parámetro es el sistema que contiene la cola de datos.
      String system           = parameters[0];

             // Cree un objeto AS400 para el servidor que tiene la cola de
datos.
                AS400 as400 = new AS400(system);


             // Construya un formato de registro para el formato de la
entrada de la cola de datos.
             // Este formato coincide con el formato de la clase DQProducer.
Un
             // registro está formado por los elementos siguientes:
             //    - un número de cuatro bytes -- el número de cliente
             //    - un número de cuatro bytes -- el número de pieza
             //    - una serie de 20 caracteres -- la descripción de la
pieza
             //    - un número de cuatro bytes -- el número de piezas de
este pedido
```

```
            // Primero cree los tipos de datos base.
            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"

            // Construya un formato de registro y llénelo con los tipos de
datos base.
            RecordFormat dataFormat = new RecordFormat();

            dataFormat.addFieldDescription(customerNumber);
            dataFormat.addFieldDescription(partNumber);
            dataFormat.addFieldDescription(partName);
            dataFormat.addFieldDescription(quantity);

            // Cree el objeto de cola de datos que representa la cola de
datos en
            // el servidor.
            DataQueue dq = new DataQueue(as400,
"/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

           boolean Continue = true;

            // Lea la primera entrada de la cola. El valor de tiempo de
espera es
            // -1 por lo que este programa esperará una entrada
infinitamente.
               System.out.println("*** Esperando una entrada para procesar
***");

            DataQueueEntry DQData = dq.read(-1);

      while (Continue)
            {

               // Se acaba de leer una entrada de la cola. Ponga los datos
en
               // un objeto de registro para que el programa pueda acceder
a los
               // campos de los datos por nombre. El objeto de registro
también
               // convertirá los datos del formato del servidor al formato
Java.
               Record data = dataFormat.getNewRecord(DQData.getData());

               // Obtenga dos valores del registro y visualícelos.
               Integer amountOrdered = (Integer) data.getField("QUANTITY");
               String  partOrdered   = (String)
data.getField("PART_NAME");

               System.out.println("Se necesita " + amountOrdered + " de " +
partOrdered);
              System.out.println(" ");
```

```
                System.out.println("*** Esperando una entrada para procesar
***");

                // Espere la entrada siguiente.
                DQData = dq.read(-1);
             }
          }
         catch (Exception e)
          {
             // Si alguna de las operaciones anteriores ha fallado, indique
que la
             // operación de la cola de datos ha fallado y envíe a la salida
la excepción.
             System.out.println("La operación de cola de datos ha fallado");
         System.out.println(e);
          }
       }

         // Visualice texto de ayuda si los parámetros son incorrectos.
           else
       {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println(" DQConsumerExample
sistema");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println(" sistema = Servidor que
tiene la cola de datos");
System.out.println("");            System.out.println("Por ejemplo:");
System.out.println("");         System.out.println(" DQConsumerExample
mySystem");
System.out.println("");System.out.println("");      }

        System.exit(0);
    }
}
```

Ejemplos: cómo se utiliza IFSFile

Los ejemplos que hay a continuación muestran cómo se utiliza la clase IFSFile:

- **Ejemplo:** crear un directorio
- **Ejemplo:** cómo se utilizan las excepciones de IFSFile para hacer seguimiento de errores
- **Ejemplo:** listar archivos con la extensión .txt
- **Ejemplo:** cómo se utiliza el método de IFSFile listFiles() para listar el contenido de un directorio

Ejemplo: crear un directorio

```
    // Cree un objeto AS400. Este nuevo
    // directorio se creará en este
    // iSeries.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree un objeto archivo que
    // represente el directorio.
IFSFile aDirectory = new IFSFile(sys, "/mydir1/mydir2/newdir");

    // Cree el directorio.
if (aDirectory.mkdir())
   System.out.println("Crear directorio ha sido satisfactorio");

    // En caso contrario, la creación del directorio ha fallado.
     else
{
    // Si el objeto ya existe,
    // averigüe si es un directorio o un
    // archivo y luego visualice un mensaje.
   if (aDirectory.exists())
   {
      if (aDirectory.isDirectory())
         System.out.println("El directorio ya existe");
     else
         System.out.println("Ya existe un archivo con ese nombre");
   }
     else
      System.out.println("Crear directorio ha fallado");
}

    // Desconecte, puesto que ya ha terminado
    // de acceder a los archivos.
sys.disconnectService(AS400.FILE);
```

Ejemplo: cómo se utilizan las excepciones de IFSFile para hacer seguimiento de errores

Cuando se produce un error, la clase IFSFile lanza la excepción ExtendedIOException. Esta excepción contiene un código de retorno que indica la causa de la anomalía. La clase IFSFile lanza la excepción incluso cuando no la lance la clase java.io que la clase IFSFile duplica. Por ejemplo, el método delete (suprimir) de java.io.File devuelve un booleano para indicar que la operación ha sido satisfactoria o que ha fallado. El método delete de IFSFile devuelve un booleano, pero si la supresión falla, se lanza una excepción ExtendedIOException. La excepción ExtendedIOException proporciona al programa Java información detallada acerca de por qué ha fallado la supresión.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree un objeto archivo que
    // represente el archivo.
IFSFile aFile = new IFSFile(sys, "/mydir1/mydir2/myfile");

// Suprima el archivo.
                       try
{
   aFile.delete();

    // La supresión ha sido satisfactoria.
   System.out.println("Supresión satisfactoria ");
}

    // La supresión ha fallado. Obtenga el
    // código de retorno de la excepción y
    // visualice por qué ha fallado la supresión.
catch (ExtendedIOException e)
{
   int rc = e.getReturnCode();

   switch (rc)
   {
      case ExtendedIOException.FILE_IN_USE:
         System.out.println("Supresión anómala, archivo en uso ");
         break;

      case ExtendedIOException.PATH_NOT_FOUND:
         System.out.println("Supresión anómala, vía no encontrada ");
         break;

    // ...Para cada error específico del
    // que desea efectuar un seguimiento.

      default:
         System.out.println("Supresión anómala, rc = ");
         System.out.println(rc);
   }
}
```

Ejemplo: listar archivos con la extensión .txt

Ejemplo 3: el programa Java puede especificar opcionalmente criterios de coincidencia al listar los archivos del directorio. Los criterios de coincidencia reducen el número de archivos devueltos por el servidor al objeto IFSFile, y así el rendimiento aumenta. El ejemplo que hay a continuación muestra cómo se listan los archivos que tienen la extensión .txt:

```
    // Cree el objeto AS400.
AS400 system = new AS400("mySystem.myCompany.com");

    // Cree el objeto archivo.
IFSFile directory = new IFSFile(system, "/");

    // Genere una lista de todos los archivos
    // que tengan la extensión .txt
```

```
String[] names = directory.list("*.txt");

    // Visualice los nombres.
if (names != null)
  for (int i = 0; i < names.length; i++)
    System.out.println(names[i]);
     else
  System.out.println("No hay archivos .txt");
```

Ejemplo: cómo se utiliza el método de IFSFile listFiles() para listar el contenido de un directorio

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////
//
// Ejemplo de IFSListFiles. Este programa utiliza las clases de IFS
// para listar el contenido de un directorio en el servidor.
//
// Sintaxis de mandato:
//    IFSListFiles sistema directorio
//
// Por ejemplo:
//    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        = "";


      // De no haberse especificado ambos parámetros, visualizar texto de
ayuda y salir.

        if (parameters.length >= 2)
        {

            // Supongamos que el primer parámetro es el nombre de sistema
            // y el segundo parámetro es el nombre de directorio

            system = parameters[0];
            directoryName = parameters[1];

                        try
            {
                // Cree un objeto AS400 para el servidor que contiene los
archivos.

                AS400 as400 = new AS400(system);



                // Cree el objeto IFSFile para el directorio.

                IFSFile directory = new IFSFile(as400, directoryName);


                // Genere una lista de elementos IFSFile. Pase el método
```

```
listFiles,
                // el objeto de filtro de directorio y los criterios de
coincidencia
                // de búsqueda. Este método almacena en la antememoria
información de
                // atributos. Por ejemplo, cuando se hace una llamada a
isDirectory()
                // en un objeto IFSFile en la matriz de archivo devuelta en
el código siguiente,
                // no se necesita ninguna llamada al servidor.
                //
                // Sin embargo, con el uso del método listFiles, la
información
                // de atributo no se renovará automáticamente desde el
                // servidor. Esto significa que la información de atributo
puede
                // ser incoherente con la del servidor.

                IFSFile[] directoryFiles = directory.listFiles(new
MyDirectoryFilter(),"*");

                // Informe al usuario si el directorio no existe o está
vacío

                if (directoryFiles == null)
                {
                    System.out.println("El directorio no existe");
            return;
                }

                else if (directoryFiles.length == 0)
                {
                    System.out.println("El directorio está vacío");
            return;
                }

                for (int i=0; i< directoryFiles.length; i++)
                    {
                        // Imprima información en lista.
                        // Imprima el nombre del archivo actual.

                        System.out.print(directoryFiles[i].getName());


                        // Rellene la salida para que las columnas queden
alineadas

                        for (int j = directoryFiles[i].getName().length(); j
<18; j++)
                        System.out.print("   ");


                        // Imprima la fecha en que se ha modificado el
archivo por última vez.

                        long changeDate = directoryFiles[i].lastModified();
                        Date d = new Date(changeDate);
                        System.out.print(d);
                        System.out.print("   ");


                        // Imprima si la entrada es un archivo o un
```

```
directorio

                        System.out.print("   ");

                        if (directoryFiles[i].isDirectory())
            System.out.println("");
          else
                            System.out.println(directoryFiles[i].length());

                    }
            }

        catch (Exception e)
            {
                // Si alguna de las operaciones anteriores ha fallado,
indique que la
                // operación de lista ha fallado y envíe a la salida la
excepción.

                System.out.println("La operación de lista ha fallado");
        System.out.println(e);
            }
        }

        // Visualice texto de ayuda si los parámetros son incorrectos.

          else
        {
            System.out.println("");
            System.out.println("");
            System.out.println("");
            System.out.println("Los parámetros no son correctos. La sintaxis
del mandato es:");
            System.out.println("");
            System.out.println("   IFSListFiles as400 directorio");
            System.out.println("");
            System.out.println("Donde");
            System.out.println("");
            System.out.println("   as400 = sistema que contiene los
archivos");
            System.out.println("   directorio = directorio que se listará");
            System.out.println("");
            System.out.println("Por ejemplo:");
            System.out.println("");
            System.out.println("   IFSListFiles mySystem /dir1/dir2");
            System.out.println("");
            System.out.println("");
        }

       System.exit(0);
    }
}


////////////////////////////////////////////////////////////////////////
//
// La clase de filtro de directorio imprime información del objeto archivo.
//
// Otra forma de emplear el filtro es simplemente devolver true o false
// según la información del objeto archivo. Esto permite que la función
// principal decida qué hacer con la lista de archivos que cumplen los
// criterios de búsqueda.
```

```
//
////////////////////////////////////////////////////////////////////////////


class MyDirectoryFilter implements IFSFileFilter
{
        public boolean accept(IFSFile file)
    {
                            try
        {
            // Conserve esta entrada. El retorno de true indica al objeto
IFSList
            // que devuelva este archivo en la lista de entradas devueltas
al
            // método .list().

            return true;
        }

        catch (Exception e)
        {
            return false;
        }
    }
}
```

Ejemplo: cómo se utilizan las clases del sistema de archivos integrado (IFS) para copiar un archivo de un directorio en otro

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de IFSCopyFile. Este programa utiliza las clases de IFS
// para copiar un archivo de un directorio en otro del servidor.
//
// Sintaxis de mandato:
//    IFSCopyFile sistema ystem nombreOrigen nombreDestino
//
// Por ejemplo:
//    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;

      // De no haberse especificado los tres parámetros, visualizar texto de
ayuda y salir.

      if (parameters.length > 2)
      {

         // Supongamos que el primer parámetro es el nombre de sistema,
         // el segundo parámetro es el nombre del origen y
         // el tercer parámetro es el nombre del destino.

            system = parameters[0];
         sourceName = parameters[1];
         targetName = parameters[2];

                           try
         {
            // Cree un objeto AS400 para el servidor que contiene los
archivos.

                AS400 as400 = new AS400(system);
```

```
            // Abra el archivo fuente para acceso exclusivo.

            source = new IFSFileInputStream(as400,
                                            sourceName,
                                            IFSFileInputStream.SHARE_NONE);

            System.out.println("El archivo fuente se ha abierto
correctamente");



            // Abra el archivo destino para acceso exclusivo.

            target = new IFSFileOutputStream(as400,
                                             targetName,
                                          IFSFileOutputStream.SHARE_NONE,
                                          false);

            System.out.println("El archivo destino se ha abierto
correctamente");



            // Lea los primeros 64K bytes del archivo fuente.

            int bytesRead = source.read(buffer);



            // Mientras haya datos en el archivo fuente, copie los datos del
            // archivo fuente en el archivo destino.

            while (bytesRead > 0)
            {
               target.write(buffer, 0, bytesRead);
               bytesRead = source.read(buffer);
            }

            System.out.println("Los datos se han copiado correctamente");



            // Cierre los archivos fuente y destino.

            source.close();
            target.close();



            // Obtenga la fecha/hora de la última modificación del archivo
fuente y
            // establézcala en el archivo destino.

            IFSFile src = new IFSFile(as400, sourceName);
            long dateTime = src.lastModified();

            IFSFile tgt = new IFSFile(as400, targetName);
            tgt.setLastModified(dateTime);

            System.out.println("La fecha/hora se ha establecido
```

```
correctamente en el archivo destino");
            System.out.println("La operación de copia se ha efectuado
correctamente");

         }
        catch (Exception e)
         {
            // Si alguna de las operaciones anteriores ha fallado, indique
que la
            // operación de copia ha fallado y envíe a la salida la
excepción.

            System.out.println("La operación de copia ha fallado");
        System.out.println(e);
         }
      }

        // Visualice texto de ayuda si los parámetros son incorrectos.

          else
      {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println("   IFSCopyFile as400
origen destino");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println("   as400  = sistema que
contiene los archivos");
         System.out.println("   origen = archivo fuente con el formato
/vía/vía/nombre");
         System.out.println("   destino = archivo destino con el formato
/vía/vía/nombre");
System.out.println("");            System.out.println("Por ejemplo:");
System.out.println("");         System.out.println("   IFSCopyFile myAS400
/dir1/dir2/a.txt  /dir3/b.txt");
System.out.println("");System.out.println("");      }

       System.exit(0);
   }
}
```

Ejemplo: cómo se utilizan las clases del sistema de archivos integrado (IFS) para listar el contenido de un directorio

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de IFSListFile. Este programa utiliza las clases de IFS
// para listar el contenido de un directorio en el servidor.
//
// Sintaxis de mandato:
//    IFSList sistema directorio
//
// Por ejemplo:
//    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        = "";


      // De no haberse especificado ambos parámetros, visualizar texto de
ayuda y salir.

        if (parameters.length >= 2)
      {

            // Supongamos que el primer parámetro es el nombre de sistema
            // y el segundo parámetro es el nombre de directorio

            system = parameters[0];
            directoryName = parameters[1];

                        try
         {
                // Cree un objeto AS400 para el servidor que contiene los
archivos.

                AS400 as400 = new AS400(system);



                // Cree el objeto IFSFile para el directorio.

                IFSFile directory = new IFSFile(as400, directoryName);
```

```
            // Genere la lista de nombre. Pase el método de lista, el objeto
            // de filtro de directorio y los criterios de coincidencia de
búsqueda.
            //
            // Nota: este ejemplo no procesa el objeto de filtro.
            // También se puede procesar la lista antes de que vuelva
            // de la llamada del método de lista.

            String[] directoryNames = directory.list(new
MyDirectoryFilter(),"*");



                // Informe al usuario si el directorio no existe o está
vacío

            if (directoryNames == null)
                    System.out.println("El directorio no existe");

            else if (directoryNames.length == 0)
                    System.out.println("El directorio está vacío");
         }

        catch (Exception e)
         {
                // Si alguna de las operaciones anteriores ha fallado,
indique que la
                // operación de lista ha fallado y envíe a la salida la
excepción.

                System.out.println("La operación de lista ha fallado");
        System.out.println(e);
         }
      }

        // Visualice texto de ayuda si los parámetros son incorrectos.

          else
      {
            System.out.println("");
            System.out.println("");
            System.out.println("");
            System.out.println("Los parámetros no son correctos. La sintaxis
del mandato es:");
            System.out.println("");
         System.out.println("   IFSList as400 directorio");
            System.out.println("");
            System.out.println("Donde");
            System.out.println("");
            System.out.println("   as400 = sistema que contiene los
archivos");
            System.out.println("   directorio = directorio que se listará");
            System.out.println("");
            System.out.println("Por ejemplo:");
            System.out.println("");
         System.out.println("   IFSCopyFile mySystem /dir1/dir2");
            System.out.println("");
            System.out.println("");
      }

       System.exit(0);
   }
```

```
}


////////////////////////////////////////////////////////////////////////
//
// La clase de filtro de directorio imprime información del objeto archivo.
//
// Otra forma de emplear el filtro es simplemente devolver true o false
// según la información del objeto archivo. Esto permite que la función
// principal decida qué hacer con la lista de archivos que cumplen los
// criterios de búsqueda.
//
////////////////////////////////////////////////////////////////////////



class MyDirectoryFilter implements IFSFileFilter
{
        public boolean accept(IFSFile file)
   {
                            try
      {
         // Imprima el nombre del archivo actual.

         System.out.print(file.getName());



                        // Rellene la salida para que las columnas queden
alineadas

         for (int i = file.getName().length(); i < 18; i++)
                        System.out.print("   ");



                        // Imprima la fecha en que se ha modificado el
archivo por última vez.

         long changeDate = file.lastModified();
                        Date d = new Date(changeDate);
                        System.out.print(d);
                        System.out.print("   ");



                        // Imprima si la entrada es un archivo o un
directorio

                        System.out.print("   ");

         if (file.isDirectory())
            System.out.println("<DIR>");
          else
            System.out.println(file.length());



            // Conserve esta entrada. El retorno de true indica al objeto
IFSList
            // que devuelva este archivo en la lista de entradas devueltas
al
```

```
            // método .list().

            return true;
        }

          catch (Exception e)
        {
            return false;
        }
    }
}
```

Ejemplo: cómo se utiliza JDBCPopulate para crear y llenar con datos una tabla

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo JDBCPopulate. Este programa utiliza el controlador JDBC de
// IBM Toolbox para Java para crear y llenar con datos (poblar) una tabla.
//
// Sintaxis de mandato:
//    JDBCPopulate sistema nombreColección nombreTabla
//
// Por ejemplo:
//    JDBCPopulate MiSistema MiBiblioteca MiTabla
//
///////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{



    // Series que se añadirán a la columna WORD de la tabla.
    private static final String words[]
        = { "Uno",      "Dos",      "Tres",    "Cuatro",     "Cinco",
            "Seis",      "Siete",    "Ocho",    "Nueve",     "Diez",
            "Once",   "Doce",   "Trece", "Catorce", "Quince",
            "Dieciséis",  "Diecisiete","Dieciocho", "Diecinueve", "Veinte"
};



    public static void main (String[] parameters)
    {
        // Compruebe los parámetros de entrada.
        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Utilización:");
            System.out.println("");
            System.out.println("   JDBCPopulate sistema nombreColección
nombreTabla");
            System.out.println("");
            System.out.println("");
            System.out.println("Por ejemplo:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCPopulate MiSistema MiBiblioteca
MiTabla");
            System.out.println("");
            return;
        }

      String system           = parameters[0];
        String collectionName   = parameters[1];
        String tableName        = parameters[2];

        Connection connection   = null;
```

```
  try {

            // Cargue el controlador JDBC de IBM Toolbox para Java.
            DriverManager.registerDriver(new
com.ibm.as400.access.AS400JDBCDriver());

            // Obtenga una conexión con la base de datos. Debido a que no
            // proporcionamos un ID de usuario ni una contraseña, aparecerá
una solicitud.            //
            // Observe que aquí proporcionamos un esquema por omisión por lo
que
            // no es necesario calificar el nombre de tabla en las
sentencias
            // SQL.
            //
            connection = DriverManager.getConnection ("jdbc:as400://"
                + system + "/" + collectionName);
         // Elimine la tabla si ya existe.
  try {
                Statement dropTable = connection.createStatement ();
                dropTable.executeUpdate ("DROP TABLE " + tableName);
}
            catch (SQLException e) {
                // Hacer caso omiso.
            }

         // Cree la tabla.
            Statement createTable = connection.createStatement ();
            createTable.executeUpdate ("CREATE TABLE " + tableName
                + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
                + " SQUAREROOT DOUBLE)");
         // Prepare una sentencia para insertar filas. Dado que
         // la ejecutamos varias veces, es mejor emplear una
         // PreparedStatement y marcadores de parámetros.
            PreparedStatement insert = connection.prepareStatement ("INSERT
INTO "
                + tableName + " (I, WORD, SQUARE, SQUAREROOT) "
                + " VALUES (?, ?, ?, ?)");
         // Llene con datos la tabla.
            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 ();            }

         // Envíe a la salida un mensaje de finalización.
            System.out.println ("La tabla " + collectionName + "." +
tableName
                + " se ha llenado con datos.");
        }

    catch (Exception e) {
         System.out.println ();
            System.out.println ("ERROR: " + e.getMessage());
        }

        finally {

            // Borre.
  try {
```

```
            if (connection != null)
               connection.close ();
            }
            catch (SQLException e) {
                // Hacer caso omiso.
            }
        }

        System.exit (0);
    }


}
```

Ejemplo: cómo se utiliza JDBCQuery para consultar una tabla

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo JDBCQuery. Este programa utiliza el controlador JDBC de IBM
// Toolbox para Java para consultar una tabla y enviar su contenido a la
salida.
//
// Sintaxis de mandato:
//    JDBCQuery sistema nombreColección nombreTabla
//
// Por ejemplo:
//    JDBCQuery MiSistema qiws qcustcdt
//
////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{



    // Dé formato a una serie (String) para que tenga la anchura
especificada.
    private static String format (String s, int width)
    {
        String formattedString;

        // La serie es más corta que la anchura especificada,
        // por lo que tenemos que rellenarla con blancos.
        if (s.length() < width) {
            StringBuffer buffer = new StringBuffer (s);
            for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
            formattedString = buffer.toString();
        }

        // En el caso contrario, tendremos que truncar la serie.
          else
            formattedString = s.substring (0, width);

        return formattedString;
    }



    public static void main (String[] parameters)
    {
        // Compruebe los parámetros de entrada.
        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Utilización:");
            System.out.println("");
            System.out.println("   JDBCQuery sistema nombreColección
nombreTabla");
            System.out.println("");
            System.out.println("");
```

```
            System.out.println("Por ejemplo:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCQuery MiSistema qiws qcustcdt");
            System.out.println("");
            return;
          }

        String system           = parameters[0];
          String collectionName   = parameters[1];
          String tableName        = parameters[2];

          Connection connection   = null;

    try {

            // Cargue el controlador JDBC de IBM Toolbox para Java.
            DriverManager.registerDriver(new
com.ibm.as400.access.AS400JDBCDriver());

            // Obtenga una conexión con la base de datos. Debido a que no
            // proporcionamos un ID de usuario ni una contraseña, aparecerá
una solicitud.            connection = DriverManager.getConnection
("jdbc:as400://" + system);
            DatabaseMetaData dmd = connection.getMetaData ();
            // Ejecute la consulta.            Statement select =
connection.createStatement ();
            ResultSet rs = select.executeQuery ("SELECT * FROM "
                + collectionName + dmd.getCatalogSeparator() + tableName);
            // Obtenga información sobre el conjunto de resultados.
Establezca que la anchura
            // de la columna sea la longitud mayor de las dos: la longitud
de la etiqueta
            // o la longitud de los datos.            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));           }

            // Envíe a la salida las cabeceras de columna.
                for (int i = 1; i <= columnCount; ++i) {
                System.out.print (format (rsmd.getColumnLabel(i),
columnWidths[i-1]));
                    System.out.print (" ");
            }
         System.out.println ();

            // Envíe a la salida una línea de guiones.
            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 ();

            // Itere a través de las filas del conjunto de resultados y
envíe a la
```

```
            // salida las columnas que hay en cada fila.             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 {

            // Borre.
  try {
            if (connection != null)
               connection.close ();
            }
            catch (SQLException e) {
                // Hacer caso omiso.
            }
        }

        System.exit (0);
    }



}
```

Ejemplo: cómo se utiliza JobList para listar información de identificación de trabajo

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Este programa es un ejemplo del soporte de trabajos de IBM Toolbox
// para Java. Lista información de identificación de trabajo para un
// usuario específico del sistema.
//
// Sintaxis de mandato:
//    listJobs2 sistema IDUsuario contraseña
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;

public class listJobs2 extends Object
{

                    // Cree un objeto por si queremos hacer
                    // llamadas a métodos no estáticos.
  public static void main(String[] parameters)
   {
      listJobs2 me = new listJobs2();
      me.Main(parameters);
       System.exit(0);
   }



   void Main(String[] parameters)
   {

         // De no haberse especificado un sistema, visualizar texto de ayuda
y salir.
      if (parameters.length == 0)
      {
         showHelp();
            return;
      }

                    // Asigne los parámetros a las variables. El
                    // primer parámetro se supone que es el nombre
                    // del sistema, el segundo un ID de usuario
                    // y el tercero una contraseña.
      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
      {

                     // Cree un objeto AS400 con el nombre de sistema
                     // especificado por el usuario. Establezca el ID de
                     // usuario y la contraseña si el usuario los especifica.
            AS400 as400 = new AS400(parameters[0]);

            if (userID != null)
            as400.setUserId(userID);

         if (password != null)
            as400.setPassword(password);



         System.out.println("recuperando lista...     ");


                     // Cree un objeto jobList. Este objeto se emplea para
                     // recuperar la lista de trabajos activos en el
servidor.
         JobList jobList = new JobList(as400);


         // Obtenga la lista de trabajos activos.
     Enumeration list = jobList.getJobs();


                     // Para cada trabajo de la lista...
     while (list.hasMoreElements())
         {

                     // Obtenga un trabajo de la lista. Si se ha especificado
                     // un ID de usuario imprima información de
identificación
                     // sólo si el usuario del trabajo coincide con el ID de
                     // usuario. Si no se ha especificado un ID de usuario,
                     // imprima información para todos los trabajos del
sistema.
            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("Error inesperado");
      e.printStackTrace();
      }
   }



        // Visualice texto de ayuda si los parámetros son incorrectos.
   void showHelp()
   {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println("   listJobs2 sistema
IDUsuario contraseña");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println("   System   = servidor
al que debe conectarse");
         System.out.println("   IDUsuario   = ID de usuario válido en ese
sistema ");
         System.out.println("   contraseña = contraseña para el ID de
usuario (opcional)");
System.out.println("");           System.out.println("Por ejemplo:");
System.out.println("");         System.out.println("   listJobs2 MYAS400
JavaUser pwd1");
System.out.println("");System.out.println("");   }
}
```

Ejemplo: cómo se utiliza JobList para obtener una lista de trabajos

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Este programa es un ejemplo de las clases de "trabajo" de IBM
// Toolbox para Java. Obtiene una lista de trabajos existentes en el
servidor y
// envía a la salida el estado del trabajo seguido del identificador de
trabajo.
//
//
// Sintaxis de mandato:
//    listJobs sistema IDUsuario contraseña
//
// (El ID de usuario y la contraseña son opcionales)
//
///////////////////////////////////////////////////////////////////////

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)
   {

        // De no haberse especificado un sistema, visualizar texto de ayuda
y salir.
      if (parameters.length == 0)
      {
         showHelp();
            return;
      }

      // Configure los parámetros de objeto AS400. El primero es el nombre
del
      // sistema y debe especificarlo el usuario. El segundo y el tercero
son
      // opcionales (ID de usuario y contraseña). Convierta el ID de usuario
y la
      // contraseña a mayúsculas antes de establecerlos en el objeto AS400.
      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
      {

         // Cree un objeto AS400 con el nombre de sistema especificado por
el usuario.
            AS400 as400 = new AS400(parameters[0]);

         // Si se ha especificado un ID de usuario o una contraseña,
establézcalos
         // en el objeto AS400.
            if (userID != null)
            as400.setUserId(userID);

         if (password != null)
            as400.setPassword(password);

         // Cree un objeto de lista de trabajos. El parámetro de entrada es
el AS400
         // del que queremos obtener información de trabajo.
         JobList jobList = new JobList(as400);

         // Obtenga una lista de los trabajos que se ejecutan en el
servidor.
         Enumeration listOfJobs = jobList.getJobs();

         // Para cada trabajo de la lista imprima información sobre el
trabajo.
         while (listOfJobs.hasMoreElements())
         {
            printJobInfo((Job) listOfJobs.nextElement(), as400);
         }

      }
        catch (Exception e)
      {
         System.out.println("Error inesperado");
        System.out.println(e);
      }
   }



   void printJobInfo(Job job, AS400 as400)
   {

      // Cree los diversos conversores necesarios
      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);
```

```
      // Tenemos el nombre/número/etc. del trabajo a partir de la petición
de lista.
      // Ahora cree una llamada de API del servidor para obtener el estado
del trabajo.
                              try
      {
         // Cree un objeto de llamada a programa
         ProgramCall pgm = new ProgramCall(as400);

         // El programa del servidor al que llamamos tiene 5 parámetros.
            ProgramParameter[] parmlist = new ProgramParameter[5];

         // El primer parámetro es una matriz de bytes que contiene los
datos
         // de salida. Asignaremos 1k de almacenamiento intermedio para
ellos.
         parmlist[0] = new ProgramParameter( 1024 );

         // El segundo es el tamaño del almacenamiento intermedio de datos
de salida (1K).
         Integer iStatusLength = new Integer( 1024 );
         byte[]  statusLength = bin4Converter.toBytes( iStatusLength );
            parmlist[1] = new ProgramParameter( statusLength );

         // El tercero es el nombre del formato de los datos.
         // Usaremos el formato JOBI0200 ya que tiene el estado del trabajo.
         byte[] statusFormat = text8Converter.toBytes("JOBI0200");
            parmlist[2] = new ProgramParameter( statusFormat );

         // El cuarto es el nombre de trabajo con el formato "nombre usuario
número".
         // El nombre debe tener 10 caracteres, el usuario debe tener 10
caracteres y el
         // número debe tener 6 caracteres. Usaremos un conversor de texto
         // para la conversión y el relleno.
         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 );

         // El último parámetro es el identificador de trabajo. Lo dejaremos
en blanco.
         byte[] jobID = text16Converter.toBytes("                ");
         parmlist[4] = new ProgramParameter( jobID );


         // Ejecute el programa.
         if (pgm.run( "/QSYS.LIB/QUSRJOBI.PGM", parmlist )==false)
         {
            // Si el programa ha fallado, visualice el mensaje de error.
            AS400Message[] msgList = pgm.getMessageList();
            System.out.println(msgList[0].getText());
```

```
         }
          else
         {
            // De lo contrario el programa ha funcionado. Envíe a la salida
el estado seguido de
            // nombreTrabajo.Usuario.IDTrabajo
               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);
      }

   }



        // Visualice texto de ayuda si los parámetros son incorrectos.
   void showHelp()
   {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println("   listJobs sistema
IDUsuario contraseña");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println("   System   = servidor
al que debe conectarse");
         System.out.println("   IDUsuario   = ID de usuario válido en ese
sistema (opcional)");
         System.out.println("   contraseña = contraseña para el ID de
usuario (opcional)");
System.out.println("");           System.out.println("Por ejemplo:");
System.out.println("");         System.out.println("   listJobs MYAS400
JavaUser pwd1");
System.out.println("");System.out.println("");   }
}
```

Ejemplo: cómo se utiliza JobLog para visualizar los mensajes de las anotaciones de trabajo

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Este programa es un ejemplo de la función de anotaciones de trabajo de
IBM
// Toolbox para Java. Visualizará los mensajes de las anotaciones de trabajo
// de un trabajo que pertenezca al usuario actual.
//
// Sintaxis de mandato:
//    jobLogExample sistema IDUsuario contraseña
//
// (La contraseña es opcional)
//
///////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;

public class jobLogExample
{


    public static void main (String[] args)
    {
        // De no haberse especificado un sistema y un usuario, visualizar
texto de ayuda y salir.
        if (args.length < 2)
        {
            System.out.println("Utilización:  jobLogExample sistema
IDusuario <contraseña>");
            return;
        }

      String userID   = null;

                           try
        {
            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos. Si se ha
            // pasado un ID de usuario y una contraseña en la línea de
            // mandatos, establézcalos también.
            AS400 system = new AS400 (args[0]);

if (args.length > 1)            {
               userID = args[1];
               system.setUserId(userID);
            }

            if (args.length > 2)
               system.setPassword(args[2]);


            // Cree un objeto de lista de trabajos. Este objeto se usará
```

```
            // para obtener la lista de trabajos activos del sistema. Una
            // vez recuperada la lista de trabajos, el programa localizará
            // un trabajo para el usuario actual.
            JobList jobList = new JobList(system);

            // Obtenga la lista de trabajos activos en el AS/400
     Enumeration list = jobList.getJobs();

            boolean Continue = true;

            // Busque en la lista para localizar un trabajo para el usuario
actual.
            while (list.hasMoreElements() && Continue)
            {
            Job j = (Job) list.nextElement();

        if (j.getUser().trim().equalsIgnoreCase(userID))
               {
                  // Se ha encontrado un trabajo correspondiente al usuario
actual.
                  // Cree un objeto de anotaciones de trabajo para este
trabajo.
           JobLog jlog = new JobLog(system,
                                    j.getName(),
                                    j.getUser(),
                                   j.getNumber());

                  // Enumere los mensajes de las anotaciones de trabajo e
imprímalos.
           Enumeration messageList = jlog.getMessages();

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

                  // Se ha encontrado un trabajo correspondiente al usuario
actual; por lo tanto, salga.
                  Continue = false;
               }
            }
        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
        }

       System.exit(0);
    }

}
```

Ejemplo: crear archivos en spool

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo que muestra cómo crear un archivo en spool en un servidor a
partir de una corriente de entrada.
//
///////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;

import com.ibm.as400.access.*;

class NPExampleCreateSplf
{

// Método para crear el archivo en spool en el sistema especificado y
// en la cola de salida especificada a partir de la corriente de entrada
dada.
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();

        // Cree un objeto PrintParameterList con los valores que deseamos
        // modificar del archivo de impresora por omisión... Modificaremos
        // la cola de salida y el valor de copias.
        parms.setParameter(PrintObject.ATTR_COPIES, 4);
        if (outputQueue != null)
        {
           parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE,
outputQueue.getPath());
        }
        out = new SpooledFileOutputStream(system,
                                          parms,
                                          null,
                                          null);

        // Lea los datos de la corriente de entrada hasta el final de la
corriente, pasando todos los datos
        // a la corriente de salida del archivo en spool.
            do
        {
            bytesRead = in.read(buf);
            if (bytesRead != -1)
            {
                out.write(buf);
            }
```

```
        } while (bytesRead != -1);

        out.close();  // Cierre el archivo en spool

        spooledFile = out.getSpooledFile();   // Obtenga una referencia al
nuevo archivo en spool

    }
        catch (Exception e)
    {
        // Maneje la excepción...
    }
    return spooledFile;
}

}
```

Ejemplo: crear archivos en spool SCS

Este ejemplo utiliza la clase SCS3812Writer para generar una corriente de datos SCS y escribirla en un archivo en spool del servidor.

Esta aplicación puede tomar los argumentos indicados más abajo o utilizar los valores definidos por omisión:

- Nombre del servidor que recibirá el archivo en spool.
- Nombre de la cola de salida del servidor que recibirá el archivo en spool.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de "SCS3812Writer" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////

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;

            // Procese los argumentos.
            if (args.length >= 1)
            {
                system = new AS400(args[0]);    // Cree un objeto AS400
            } else {
                system = new AS400(DEFAULT_SYSTEM);
            }

            if (args.length >= 2)               // Establezca la cola de
salida
            {
                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);
```

```
            // Escriba el contenido del archivo en spool.
            scsWtr.setLeftMargin(1.0);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_5);
            scsWtr.write("          Impresión en Java");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setCPI(10);
            scsWtr.write("Este documento se ha creado con IBM Toolbox para
Java.");
            scsWtr.newLine();
            scsWtr.write("El resto de este documento muestra algunas tareas
que");
            scsWtr.newLine();
            scsWtr.write("pueden realizarse con la clase SCS3812Writer.");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setUnderline(true); scsWtr.write("Estableciendo fonts:");
scsWtr.setUnderline(false);
            scsWtr.newLine();
            scsWtr.setFont(scsWtr.FONT_COURIER_10); scsWtr.write("Font
Courier ");
            scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_10); scsWtr.write(" Font
Courier negrita ");
            scsWtr.setFont(scsWtr.FONT_COURIER_ITALIC_10); scsWtr.write("
Font Courier cursiva ");
            scsWtr.newLine();
            scsWtr.setBold(true); scsWtr.write("Font Courier negrita cursiva
");
            scsWtr.setBold(false);
            scsWtr.setCPI(10);
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setUnderline(true); scsWtr.write("Líneas por pulgada:");
scsWtr.setUnderline(false);
            scsWtr.newLine();
            scsWtr.write("Las líneas siguientes deben imprimirse a 8 líneas
por pulgada.");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setLPI(8);
            scsWtr.write("Línea uno"); scsWtr.newLine();
            scsWtr.write("Línea dos"); scsWtr.newLine();
            scsWtr.write("Línea tres"); scsWtr.newLine();
            scsWtr.write("Línea cuatro"); scsWtr.newLine();
            scsWtr.write("Línea cinco"); scsWtr.newLine();
            scsWtr.write("Línea seis"); scsWtr.newLine();
            scsWtr.write("Línea siete"); scsWtr.newLine();
            scsWtr.write("Línea ocho"); scsWtr.newLine();
            scsWtr.endPage();
            scsWtr.setLPI(6);
            scsWtr.setSourceDrawer(1);
            scsWtr.setTextOrientation(0);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.write("Esta página debe imprimirse con orientación
vertical de la bandeja 1.");
            scsWtr.endPage();
            scsWtr.setSourceDrawer(2);
            scsWtr.setTextOrientation(90);
            scsWtr.absoluteVerticalPosition(6);
```

```
            scsWtr.write("Esta página debe imprimirse con orientación
horizontal de la bandeja 2.");
            scsWtr.endPage();
            scsWtr.close();
            System.out.println("Se ha creado el archivo en spool de
ejemplo.");
       System.exit(0);
        }
        catch (Exception e)
        {
            // Maneje el error.
            System.out.println("Se ha producido una excepción al crear el
archivo en spool." + e);
       System.exit(0);
        }
    }
}
```

Ejemplo: leer archivos en spool

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo que lee un archivo en spool existente en el servidor.
//
// Este fuente es un ejemplo de "PrintObjectInputStream" de IBM Toolbox para
Java.
//
///////////////////////////////////////////////////////////////////////
        try{
        byte[] buf = new byte[2048];
        int bytesRead;
        AS400 sys = new AS400();
        SpooledFile splf = new SpooledFile( sys,          // AS400
                                            "MICR",       // nombre de
archivo en spool
                                            17,           // número de
archivo en spool
                                            "QPRTJOB",    // nombre de
trabajo
                                            "QUSER",      // usuario del
trabajo
                                            "020791" );   // número de
trabajo

        // Abra el archivo en spool para lectura y obtenga la corriente de
entrada
        // para leer de él.
        InputStream in = splf.getInputStream(null);

            do
        {
            // Lea hasta longitud_alm_intermedio bytes de datos de spool
puros
            // en el almacenamiento intermedio. Se devolverán los bytes
reales leídos.
            // Los datos serán una corriente de datos binarios de impresora
que son
            // el contenido del archivo en spool.
            bytesRead = in.read( buf );
            if( bytesRead != -1 )
            {
                // Procese los datos del archivo en spool.
               System.out.println( "Se han leído " + bytesRead + " bytes" );
            }
        } while( bytesRead != -1 );

        in.close();
    }
        catch (Exception e)
    {
        // Excepción
    }
```

Ejemplo: listar archivos en spool asíncronamente (utilizando escuchadores)

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo que muestra cómo listar todos los archivos en spool de un sistema
asíncronamente usando
// la interfaz PrintObjectListListener para obtener información de retorno a
medida que se construye
// la lista. Listar asíncronamente permite al llamador empezar a procesar
los objetos de la lista
// antes de que se construya toda la lista de modo que el usuario obtenga un
tiempo de respuesta
// más rápido.
//
///////////////////////////////////////////////////////////////////////

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;
    }

    // Liste todos los archivos en spool del sistema asíncronamente usando
un escuchador
    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(" Se están recibiendo todos los archivos en
spool asíncronamente usando un escuchador");

            SpooledFileList splfList = new SpooledFileList(system_);

            // Establezca filtros, todos los usuarios, en todas las colas
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

            // Añada el escuchador.
            splfList.addPrintObjectListListener(this);

            // Abra la lista, openAsynchronously vuelve de inmediato
            splfList.openAsynchronously();

            do
            {
                // Espere a que la lista tenga al menos 25 objetos o a que
se complete
                waitForWakeUp();

                fCompleted = splfList.isCompleted();
                size = splfList.size();

                // Envíe a la salida el nombre de todos los objetos añadidos
a la lista
                // desde la última activación
                while (listed < size)
                {
                    if (fListError)
                    {
                        System.out.println(" Excepción en lista - " +
listException);
              break;
                    }

                    if (fListClosed)
                    {
            System.out.println(" La lista se ha cerrado antes de
completarse.");
              break;
                    }

                    SpooledFile splf =
(SpooledFile)splfList.getObject(listed++);
                    if (splf != null)
                    {
                        // Envíe a la salida el nombre de este archivo en
spool
                        strSpooledFileName =
splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                        System.out.println(" archivo en spool = " +
strSpooledFileName);
                    }
                }
```

```
            } while (!fCompleted);

            // Borre una vez que haya terminado con la lista
            splfList.close();
            splfList.removePrintObjectListListener(this);
        }

        catch( ExtendedIllegalStateException e )
        {
            System.out.println(" La lista se ha cerrado antes de
completarse.");
        }

        catch (Exception e)
        {
            // Maneje las demás excepciones que existan...
      e.printStackTrace();
        }

    }

    // Aquí es donde la hebra de primer plano espera a que la hebra de fondo
    // la active cuando la lista se actualice o finalice.
    private synchronized void waitForWakeUp()
       throws InterruptedException
    {
        // No vuelva al estado en reposo si el escuchador indica que se ha
completado la lista
        if (!fListCompleted)
        {
            wait();
        }
    }

    // Los métodos siguientes implementan la interfaz
PrintObjectListListener

    // Se invoca este método cuando se cierra la lista.
    public void listClosed(PrintObjectListEvent event)
    {
        System.out.println("*****La lista se ha cerrado*****");
        fListClosed = true;
        synchronized(this)
        {
            // Establezca el distintivo para indicar que la lista
            // se ha completado y active la hebra de primer plano.
            fListCompleted = true;
                notifyAll();
        }
    }

    // Se invoca este método cuando se ha completado la lista.
    public void listCompleted(PrintObjectListEvent event)
    {
        System.out.println("*****La lista se ha completado*****");
            synchronized (this)
        {
            // Establezca el distintivo para indicar que la lista
            // se ha completado y active la hebra de primer plano.
            fListCompleted = true;
```

```
                notifyAll();
            }
        }

        // Se invoca este método cuando se produce un error al recuperar
        // la lista.
        public void listErrorOccurred(PrintObjectListEvent event)
        {
            System.out.println("*****La lista contiene un error*****");
            fListError = true;
            listException = event.getException();
            synchronized(this)
            {
                // Establezca el distintivo para indicar que la lista
                // se ha completado y active la hebra de primer plano.
                fListCompleted = true;
                    notifyAll();
            }
        }

        // Se invoca este método cuando se abre la lista.
        public void listOpened(PrintObjectListEvent event)
        {
            System.out.println("*****Se ha abierto la lista*****");
            listObjectCount = 0;
        }

        // Se invoca este método cuando se añade un objeto a la lista.
        public void listObjectAdded(PrintObjectListEvent event)
        {
            // Cada 25 objetos activaremos la hebra de primer plano
            // para obtener los objetos más recientes...
            if( (++listObjectCount % 25) == 0 )
            {
                System.out.println("*****Se han añadido 25 objetos más a la
lista*****");
                synchronized (this)
                {
                    // Active la hebra de primer plano
                    notifyAll();
                }
            }
        }

     public static void main(String args[])
        {
            NPExampleListSplfAsynch list = new NPExampleListSplfAsynch(new
AS400());
            try{
                list.listSpooledFiles();
            }
            catch (Exception e)
            {
          e.printStackTrace();
            }
           System.exit(0);
        }

    }
```

Ejemplo: listar archivos en spool asíncronamente (sin utilizar escuchadores)

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo que muestra cómo listar todos los archivos en spool de un sistema
asíncronamente
// sin usar la interfaz PrintObjectListListener. Tras abrir la lista, el
llamador
// puede realizar algún trabajo adicional antes de esperar a que se complete
la lista.
//
///////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de "PrintObjectList" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////

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;
    }

    // Liste todos los archivos en spool del sistema asíncronamente
    public void listSpooledFiles()
    {
                            try
        {
            String strSpooledFileName;
            int listed, size;

      if (system_ == null)
            {
        system_ = new AS400();
            }

            System.out.println(" Se están recibiendo todos los archivos en
spool asíncronamente sin usar un escuchador");

            SpooledFileList splfList = new SpooledFileList(system_);

            // Establezca filtros, todos los usuarios, en todas las colas
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

            // Abra la lista, openAsynchronously() vuelve inmediatamente
```

```
            // No hemos añadido ningún escuchador...
            splfList.openAsynchronously();

            System.out.println(" Realice algún proceso antes de
esperar...");

            // Realice algún proceso aquí mientras se construye la lista...

            System.out.println(" Ahora espere a que se complete la lista.");

            // Espere a que se complete la lista
            splfList.waitForListToComplete();

            Enumeration enum = splfList.getObjects();

            // Envíe a la salida el nombre de todos los objetos de la lista
      while (enum.hasMoreElements())
            {
                SpooledFile splf = (SpooledFile)enum.nextElement();
                    if (splf != null)
                {
                    // Envíe a la salida el nombre de este archivo en spool
                        strSpooledFileName =
splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                        System.out.println(" archivo en spool = " +
strSpooledFileName);
                }
            }
            // Borre una vez que haya terminado con la lista
            splfList.close();
        }

        catch (Exception e)
        {
            // Maneje las excepciones...
      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);
    }
}
```

Ejemplo: listar archivos en spool síncronamente

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo que muestra cómo se listan todos los archivos en spool de un
sistema síncronamente.
// Listar síncronamente no vuelve al llamador hasta que se construye la
lista completa.
// El usuario obtiene un tiempo de respuesta más lento que al listar
asíncronamente.
//
///////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de "PrintObjectList" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////

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(" Se están recibiendo todos los archivos en
spool síncronamente");

            SpooledFileList splfList = new SpooledFileList( system_ );

            // Establezca filtros, todos los usuarios, en todas las colas
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

            // Abra la lista, openSynchronously() vuelve cuando se completa
la lista.
            splfList.openSynchronously();
            Enumeration enum = splfList.getObjects();
```

```
      while (enum.hasMoreElements())
            {
                SpooledFile splf = (SpooledFile)enum.nextElement();
                if ( splf != null )
                {
                    // Envíe a la salida el nombre de este archivo en spool
                        strSpooledFileName =
splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                        System.out.println(" archivo en spool = " +
strSpooledFileName);
                }
            }
            // Borre una vez que haya terminado con la lista
            splfList.close();
        }
        catch (Exception e)
        {
            // Maneje las excepciones...
      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);
    }
}
```

Ejemplo: cómo se utiliza ProgramCall

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de llamada a programa. Este programa llama al programa QWCRSSTS
del servidor
// para recuperar el estado del sistema.
//
// Sintaxis de mandato:
//    PCSystemStatusExample sistema
//
// Este fuente es un ejemplo de "ProgramCall" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////




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( " " );



      // De no haberse especificado un sistema, visualizar texto de ayuda y
salir.

      if (parameters.length >= 1)
      {

                           try
         {
            // Cree un objeto AS400 para el servidor que contiene el
programa.
            // Supongamos que el primer parámetro es el nombre de sistema.

            AS400 as400 = new AS400(parameters[0]);



            // Cree la vía de acceso al programa.

            QSYSObjectPathName programName = new QSYSObjectPathName("QSYS",

"QWCRSSTS",
                                                                    "PGM");
```

```
            // Cree el objeto de llamada a programa. Asocie el objeto al
            // objeto AS400 que representa el servidor del que se obtiene el
estado.

            ProgramCall getSystemStatus = new ProgramCall(as400);



            // Cree la lista de parámetros del programa. Este programa tiene
cinco
            // parámetros que se añadirán a esta lista.

            ProgramParameter[] parmlist = new ProgramParameter[5];



            // El programa del servidor devuelve datos en el parámetro 1. Es
un parámetro
            // de salida. Asigne 64 bytes para este parámetro.

            parmlist[0] = new ProgramParameter( 64 );



            // El parámetro 2 es el tamaño de almacenamiento intermedio del
parámetro 1. Es
            // un parámetro de entrada numérico. Establezca su valor en 64,
conviértalo al
            // formato del servidor y añada el parámetro a la lista de
parámetros.

            AS400Bin4 bin4 = new AS400Bin4( );
            Integer iStatusLength = new Integer( 64 );
            byte[]  statusLength = bin4.toBytes( iStatusLength );
            parmlist[1] = new ProgramParameter( statusLength );



            // El parámetro 3 es el parámetro de formato de estado. Es un
parámetro
            // de entrada de tipo serie. Establezca el valor de serie,
conviértalo al
            // formato del servidor y añada el parámetro a la lista de
parámetros.

            AS400Text text1 = new AS400Text(8, as400);
            byte[] statusFormat = text1.toBytes("SSTS0200");
            parmlist[2] = new ProgramParameter( statusFormat );



            // El parámetro 4 es el parámetro de estadísticas de
restablecimiento. Es un parámetro
            // de entrada de tipo serie. Establezca el valor de serie,
conviértalo al
            // formato del servidor y añada el parámetro a la lista de
parámetros.

            AS400Text text3 = new AS400Text(10, as400);
```

```
            byte[] resetStats = text3.toBytes("*NO       ");
            parmlist[3] = new ProgramParameter( resetStats );



            // El parámetro 5 es el parámetro de información de error. Es un
parámetro
            // de entrada/salida. Añádalo a la lista de parámetros.

            byte[] errorInfo = new byte[32];
            parmlist[4] = new ProgramParameter( errorInfo, 0 );



            // Establezca el programa al que se llamará y la lista de
parámetros para
            // el objeto de llamada a programa.

            getSystemStatus.setProgram(programName.getPath(), parmlist );



            // Ejecute el programa y establézcalo en reposo. Ejecutamos dos
veces el programa
            // porque el primer conjunto de resultados está inflado. Si
descartamos el primer
            // conjunto de resultados y volvemos a ejecutar el mandato cinco
segundos después,
            // el número será más preciso.

            getSystemStatus.run();
            Thread.sleep(5000);



            // Ejecute el programa.

            if (getSystemStatus.run()!=true)
            {

               // Si no se ejecuta el programa, obtenga la lista de mensajes
de error
               // del objeto de programa y visualice los mensajes. El error
sería
               // similar a programa no encontrado o usuario no autorizado
para
               // el programa.

               AS400Message[] msgList = getSystemStatus.getMessageList();

               System.out.println("No se ha ejecutado el programa. Mensajes
del servidor:");

               for (int i=0; i<msgList.length; i++)
               {
                   System.out.println(msgList[i].getText());
               }
            }
```

```
            // De lo contrario, el programa se ha ejecutado.

          else
            {

               // Cree un conversor numérico de servidor a Java. Este
conversor se
               // usará en la sección siguiente para convertir la salida
               // numérica del formato de servidor al formato Java.

               AS400Bin4 as400Int = new AS400Bin4( );



               // Obtenga los resultados del programa. Los datos de salida
               // están en una matriz de bytes en el primer parámetro.

               byte[] as400Data = parmlist[0].getOutputData();



               // La utilización de la CPU es un campo numérico que empieza
en
               // el byte 32 del almacenamiento intermedio de salida.
Convierta este
               // número del formato de servidor al formato Java y envíe el
número a la salida.

               Integer cpuUtil = (Integer)as400Int.toObject( as400Data, 32
);
               cpuUtil = new Integer(cpuUtil.intValue()/10);
               System.out.print("Utilización de CPU:  ");
               System.out.print(cpuUtil);
               System.out.println("%");



               // La utilización de DASD es un campo numérico que empieza en
               // el byte 52 del almacenamiento intermedio de salida.
Convierta este
               // número del formato de servidor al formato Java y envíe el
número a la salida.

               Integer dasdUtil = (Integer)as400Int.toObject( as400Data, 52
);
               dasdUtil = new Integer(dasdUtil.intValue()/10000);
               System.out.print("Utilización de DASD: ");
               System.out.print(dasdUtil);
               System.out.println("%");



               // El número de trabajos es un campo numérico que empieza en
               // el byte 36 del almacenamiento intermedio de salida.
Convierta este
               // número del formato de servidor al formato Java y envíe el
número a la salida.

               Integer nj = (Integer)as400Int.toObject( as400Data, 36 );
               System.out.print("Trabajos activos:      ");
```

```
               System.out.println(nj);

            }

               // Este programa ha terminado de ejecutar el programa;
desconecte
               // del servidor de mandatos en el servidor. La llamada a
programa y
               // la llamada a mandato utilizan el mismo servidor en el
servidor.

            as400.disconnectService(AS400.COMMAND);
         }
        catch (Exception e)
         {
            // Si alguna de las operaciones anteriores ha fallado, indique
que el
            // programa ha fallado y envíe a la salida la excepción.

            System.out.println("La llamada a programa ha fallado");
        System.out.println(e);
         }
      }

        // Visualice texto de ayuda si los parámetros son incorrectos.

          else
       {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println("   PCSystemStatusExample
myServer");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println("   myServer = obtener el
estado de este servidor ");
System.out.println("");            System.out.println("Por ejemplo:");
System.out.println("");         System.out.println("   PCSystemStatusExample
mySystem");
System.out.println("");System.out.println("");      }

       System.exit(0);
   }
}
```

Ejemplo: cómo se utilizan las clases de acceso a nivel de registro

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo de acceso a nivel de registro (RLA). Este programa solicitará
// al usuario el nombre del servidor y el archivo que se ha de visualizar.
El
// archivo debe existir y ha de contener registros. Cada registro del
archivo
// se visualizará en la salida del sistema (System.out).
//
// Sintaxis de la llamada: java RLSequentialAccessExample
//
// Este fuente es un ejemplo de "RecordLevelAccess" de IBM Toolbox para
Java.
//
///////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLSequentialAccessExample
{
   public static void main(String[] parmeters)
   {
    // Se ha creado un lector para obtener entrada del usuario
      BufferedReader inputStream = new BufferedReader(new
InputStreamReader(System.in),1);

      // Declare variables para nombre de sistema, biblioteca, archivo y
miembro
      String systemName = "";
    String library = "";
      String file = "";
      String member = "";

      // Obtenga el nombre del sistema y el archivo que se visualizará del
usuario
    System.out.println();
                            try
      {
         System.out.print("Nombre del sistema: ");
        systemName = inputStream.readLine();

        System.out.print("Biblioteca en la que existe el archivo: ");
        library = inputStream.readLine();

        System.out.print("Nombre de archivo: ");
        file = inputStream.readLine();

        System.out.print("Nombre de miembro (pulse Intro para el primer
miembro): ");
        member = inputStream.readLine();
        if (member.equals(""))
        {
```

```
          member = "*FIRST";
        }

    System.out.println();
      }
        catch (Exception e)
      {
        System.out.println("Error al obtener entrada de usuario.");
      e.printStackTrace();
       System.exit(0);
      }

      // Cree un objeto AS400 y realice la conexión para el servicio de
acceso a nivel de registro.
      AS400 system = new AS400(systemName);
                           try
      {
        system.connectService(AS400.RECORDACCESS);
      }
      catch(Exception e)
      {
      System.out.println("No ha sido posible conectar para acceso a nivel de
registro.");
        System.out.println("Vea si en el archivo readme hay instrucciones
especiales relacionadas con el acceso a nivel de registro");
      e.printStackTrace();
       System.exit(0);
      }

      // Cree un objeto QSYSObjectPathName para obtener el formato del
nombre de vía del sistema de archivos integrado
      // del archivo que se visualizará.
      QSYSObjectPathName filePathName = new QSYSObjectPathName(library,
file, member, "MBR");
      // Cree un objeto SequentialFile que represente el archivo que se
visualizará.
      SequentialFile theFile = new SequentialFile(system,
filePathName.getPath());
         // Recupere el formato de registro correspondiente al archivo.
      AS400FileRecordDescription recordDescription = new
AS400FileRecordDescription(system, filePathName.getPath());
                           try
      {
        RecordFormat[] format = recordDescription.retrieveRecordFormat();
         // Establezca el formato de registro del archivo
        theFile.setRecordFormat(format[0]);
        // Abra el archivo para lectura. Lea 100 registros en una sola vez
si es posible.
        theFile.open(AS400File.READ_ONLY, 100,
AS400File.COMMIT_LOCK_LEVEL_NONE);
        // Visualice cada uno de los registros del archivo.
        System.out.println("Se va a visualizar el archivo " +
library.toUpperCase() + "/" + file.toUpperCase() + "(" +
theFile.getMemberName().trim() + "):");

        Record record = theFile.readNext();     while(record != null)
        {
          System.out.println(record);
          record = theFile.readNext();
        }
    System.out.println();
```

```
         // Cierre el archivo.
        theFile.close();

        // Desconecte del servicio de acceso a nivel de registro.
    system.disconnectService(AS400.RECORDACCESS);
      }
        catch (Exception e)
      {
        System.out.println("Se produjo un error al intentar visualizar el
archivo.");
      e.printStackTrace();

                            try
        {
         // Cierre el archivo.
        theFile.close();
        }
        catch(Exception x)
        {
        }

        // Desconecte del servicio de acceso a nivel de registro.
    system.disconnectService(AS400.RECORDACCESS);
       System.exit(0);
      }

      // Asegúrese de que finaliza la aplicación; en el readme encontrará
detalles
       System.exit(0);
    }
  }
```

Ejemplo: cómo se utilizan las clases de acceso a nivel de registro para leer registros de un archivo

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo de acceso a nivel de registro. Este programa utiliza las clases
de acceso
// a nivel de registro para leer registros de un archivo del servidor.
//
// Sintaxis de mandato:
//    java RLReadFile sistema
//
// Este programa lee los registros del archivo de base de datos de ejemplo
// de CA/400 (QCUSTCDT en la biblioteca QIWS). Si cambia este ejemplo para
actualizar
// registros debe hacer una copia de QCUSTCDT y actualizar la copia.
//
// Este fuente es un ejemplo de "acceso a nivel de registro" de IBM Toolbox
para Java.
//
///////////////////////////////////////////////////////////////////////////


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 = "";

      // Continúe únicamente si se ha especificado un nombre de archivo.

      if (parameters.length >= 1)
      {

                           try
         {

            // Supongamos que el primer parámetro es el nombre de sistema.

            system = parameters[0];

             // Cree un objeto AS400 para el servidor que tiene el archivo.

             AS400 as400 = new AS400(system);


            // Cree una descripción de registro para el archivo. El archivo
es QCUSTCDT
            // en la biblioteca QIWS.
```

```
            ZonedDecimalFieldDescription customerNumber =
                        new ZonedDecimalFieldDescription(new
AS400ZonedDecimal(6,0),
                                                         "NÚMCLI");
            CharacterFieldDescription lastName =
                        new CharacterFieldDescription(new AS400Text(8,
as400), "APELLIDO");

            CharacterFieldDescription initials =
                        new CharacterFieldDescription(new AS400Text(3,
as400), "INIC");

            CharacterFieldDescription street =
                        new CharacterFieldDescription(new AS400Text(13,
as400), "CALLE");

            CharacterFieldDescription city =
                        new CharacterFieldDescription(new AS400Text(6,
as400), "CIUDAD");

            CharacterFieldDescription state =
                        new CharacterFieldDescription(new AS400Text(2,
as400), "ESTADO");

            ZonedDecimalFieldDescription zipCode =
                        new ZonedDecimalFieldDescription(new
AS400ZonedDecimal(5,0),
                                                         "CÓDPOST");
            ZonedDecimalFieldDescription creditLimit =
                        new ZonedDecimalFieldDescription(new
AS400ZonedDecimal(4,0),
                                                         "LMTCDT");
            ZonedDecimalFieldDescription chargeCode =
                        new ZonedDecimalFieldDescription(new
AS400ZonedDecimal(1,0),
                                                         "CÓDCARG");
            ZonedDecimalFieldDescription balanceDue =
                        new ZonedDecimalFieldDescription(new
AS400ZonedDecimal(6,2),
                                                         "SALDO");
            ZonedDecimalFieldDescription creditDue =
                        new ZonedDecimalFieldDescription(new
AS400ZonedDecimal(6,2),
                                                         "CDTVENC");


            // El nombre de formato de registro debe especificarse para un
archivo DDM.
            // En el caso del archivo QCUSTCDT, su formato de registro se
llama 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);


            // Cree el objeto archivo secuencial que representa el
            // archivo en el servidor. Utilizaremos un objeto
QSYSObjectPathName
            // para obtener el nombre del archivo con el formato correcto.

            QSYSObjectPathName fileName = new QSYSObjectPathName("QIWS",
                                                                 "QCUSTCDT",
                                                                 "FILE");

            SequentialFile file = new SequentialFile(as400,
fileName.getPath());


            // Permita que el objeto de archivo sepa el formato de los
registros.

            file.setRecordFormat(qcustcdt);


            // Abra el archivo para acceso de sólo lectura. Especifique el
            // factor de bloques 10 (el objeto de archivo obtendrá 10
registros al
            // acceder al servidor para los datos). No utilice el control de
            // compromiso.

            file.open(SequentialFile.READ_ONLY,
                      10,
                      SequentialFile.COMMIT_LOCK_LEVEL_NONE);


            // Lea el primer registro del archivo.

            Record data = file.readNext();


            // Itere en bucle mientras haya registros en el archivo
(mientras no se haya
            // alcanzado el fin de archivo).

         while (data != null)
            {

               // Visualice el registro únicamente si el saldo es superior a
               // cero. En ese caso, visualice el nombre del cliente y el
               // saldo. El código siguiente obtiene campos del
               // registro por nombre de campo. Al recuperarse el campo
               // del registro se convierte del formato de servidor al
               // formato Java.

               if (((BigDecimal)data.getField("SALDO")).floatValue() > 0.0)
               {
                  System.out.print((String) data.getField("INIC") + "  ");
                  System.out.print((String) data.getField("APELLIDO") + "
");
```

```
                System.out.println((BigDecimal) data.getField("SALDO"));
             }

             // Lea el registro siguiente del archivo.

             data = file.readNext();
          }


          // Cuando no haya más registros para leer, desconéctese del
servidor.

          as400.disconnectAllServices();
       }

      catch (Exception e)
       {

          // Si alguna de las operaciones anteriores ha fallado, imprima
un mensaje de error
          // y envíe a la salida la excepción.

          System.out.println("No se ha podido leer el archivo");
      System.out.println(e);
       }
    }


      // Visualice texto de ayuda si los parámetros son incorrectos.

        else
    {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println("   RLReadFile as400");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println("   as400 = sistema que
contiene el archivo");
System.out.println("");            System.out.println("Por ejemplo:");
System.out.println("");         System.out.println("   RLReadFile
mySystem");
System.out.println("");System.out.println("");
System.out.println("Nota: este programa lee el archivo de base de datos
QIWS/QCUSTCDT.  ");
System.out.println("");System.out.println("");      }

     System.exit(0);

  }
}
```

Ejemplo: cómo se utilizan las clases de acceso a nivel de registro para leer registros por clave

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo de acceso a nivel de registro. Este programa utiliza las clases
de acceso
// a nivel de registro para leer registros por clave de un archivo del
servidor.
// Se pedirá al usuario el nombre del servidor para el que se hará la
ejecución
// y la biblioteca en que se creará el archivo QCUSTCDTKY.
//
// Sintaxis de mandato:
//    java RLKeyedFileExample
//
// Este programa copiará los registros del archivo de base de datos de
ejemplo
// de iSeries Access para Windows (QCUSTCDT en la biblioteca QIWS) en el
archivo QCUSTCDTKY con
// el mismo formato que QIWS/QCUSTCDT pero que tiene establecido el campo
NÚMCLI
// como clave del archivo.
//
// Este fuente es un ejemplo de "acceso a nivel de registro" de IBM Toolbox
para Java.
//
///////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.math.*;
import com.ibm.as400.access.*;

public class RLKeyedFileExample
{
  public static void main(String[] parameters)
  {

    // Se ha creado un lector para obtener entrada del usuario
      BufferedReader inputStream = new BufferedReader(new
InputStreamReader(System.in),1);

      // Declare variables para nombre de sistema, biblioteca, archivo y
miembro
    String systemName = "";
    String library = "";

    // Obtenga el nombre de sistema del usuario
    System.out.println();
                           try
    {
      System.out.print("Nombre del sistema: ");
        systemName = inputStream.readLine();
```

```
      System.out.print("Biblioteca en la que se creará el archivo
QCUSTCDTKY: ");
        library = inputStream.readLine();
    }
    catch(Exception e)
    {
      System.out.println("Error al obtener entrada de usuario.");
      e.printStackTrace();
       System.exit(0);
    }

      // Cree un objeto AS400 y realice la conexión para el servicio de
acceso a nivel de registro.
      AS400 system = new AS400(systemName);
                           try
    {
        system.connectService(AS400.RECORDACCESS);
    }
    catch(Exception e)
    {
      System.out.println("No ha sido posible conectar para acceso a nivel de
registro.");
       System.out.println("Vea si en el archivo readme hay instrucciones
especiales relacionadas con el acceso a nivel de registro");
      e.printStackTrace();
       System.exit(0);
    }

    RecordFormat qcustcdtFormat = null;
                           try
    {
      // Cree el objeto RecordFormat para crear el archivo. El formato de
registro del nuevo
      // archivo será el mismo que el del archivo QIWS/QCUSTCDT. Sin
embargo, haremos que el
      // campo NÚMCLI sea un campo clave.
      AS400FileRecordDescription recordDescription = new
AS400FileRecordDescription(system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

      // Sólo hay un formato de registro para el archivo, por lo que debe
tomar el primer elemento (el único)
      // de la matriz RecordFormat devuelta como RecordFormat para el
archivo.
      System.out.println("Se está recuperando formato de registro de
QIWS/QCUSTCDT...");
      qcustcdtFormat = recordDescription.retrieveRecordFormat()[0];
      // Indique que NÚMCLI es un campo clave
      qcustcdtFormat.addKeyFieldDescription("NÚMCLI");
    }
    catch(Exception e)
    {
      System.out.println("No ha sido posible recuperar el formato de
registro de QIWS/QCUSTCDT");
      e.printStackTrace();
       System.exit(0);
    }

    // Cree el objeto de archivo por clave que representa el
    // archivo que crearemos en el servidor. Utilizaremos un objeto
QSYSObectPathName
```

```
            // para obtener el nombre del archivo con el formato correcto.
    QSYSObjectPathName fileName = new QSYSObjectPathName(library,
                                                         "QCUSTCDTKY",
                                                         "*FILE",
                                                         "MBR");
    KeyedFile file = new KeyedFile(system, fileName.getPath());

                            try
    {
      System.out.println("Se está creando el archivo " + library +
"/QCUSTCDTKY...");
      // Cree el archivo con el objeto qcustcdtFormat
      file.create(qcustcdtFormat, "Archivo QCUSTCDT por clave");

      // Llene el archivo con los registros incluidos en QIWS/QCUSTCDT
      copyRecords(system, library);

      // Abra el archivo para acceso de sólo lectura. Dado que accederemos
al
      // archivo de forma aleatoria, especifique el factor de bloques 1. El
      // parámetro nivel de bloqueo de compromiso no se tendrá en cuenta
porque
      // no se ha iniciado el control de compromiso.
      file.open(AS400File.READ_ONLY,
                1,
                AS400File.COMMIT_LOCK_LEVEL_NONE);

      // Supongamos que deseamos visualizar la información de los clientes
      // 192837, 392859 y 938472
      // El campo NÚMCLI es un campo decimal con zona de longitud 6 sin
      // posiciones decimales. Por consiguiente, el valor del campo clave se
      // representa con un BigDecimal.
      BigDecimal[] keyValues = {new BigDecimal(192837), new
BigDecimal(392859), new BigDecimal(938472)};

      // Cree la clave para leer los registros. La clave para un objeto
KeyedFile
      // se especifica con Object[]
     Object[] key = new Object[1];

      Record data = null;
      for (int i = 0; i < keyValues.length; i++)
      {
        // Configure la clave para lectura
        key[0] = keyValues[i];

        // Lea el registro correspondiente al número de cliente keyValues[i]
        data = file.read(key);
        if (data != null)
        {
               // Visualice el registro únicamente si el saldo es superior a
               // cero. En ese caso, visualice el nombre del cliente y el
               // saldo. El código siguiente obtiene campos del
               // registro por nombre de campo. Al recuperarse el campo
               // del registro se convierte del formato de servidor al
               // formato Java.
               if (((BigDecimal)data.getField("SALDO")).floatValue() > 0.0)
          {
                  System.out.print((String) data.getField("INIC") + "  ");
                  System.out.print((String) data.getField("APELLIDO") + "
```

```
");
                    System.out.println((BigDecimal) data.getField("SALDO"));
          }
        }
      }

      // Todas las operaciones con el archivo han terminado
     file.close();

        // Elimine el archivo del sistema del usuario
        file.delete();
    }
    catch(Exception e)
    {
      System.out.println("No se ha podido crear/leer en QTEMP/QCUSTCDT");
      e.printStackTrace();
                            try
      {
     file.close();
        // Elimine el archivo del sistema del usuario
        file.delete();
      }
        catch(Exception x)
      {
      }
    }

    // Todas las operaciones con el acceso a nivel de registro han
terminado; desconecte del
    // servidor de acceso a nivel de registro.
    system.disconnectService(AS400.RECORDACCESS);
       System.exit(0);
  }

  public static void copyRecords(AS400 system, String library)
  {
    // Utilice la clase CommandCall para ejecutar el mandato CPYF para
copiar los registros
    // de QIWS/QCUSTCDT en QTEMP/QCUSTCDT
    CommandCall c = new CommandCall(system, "CPYF FROMFILE(QIWS/QCUSTCDT)
TOFILE(" + library + "/QCUSTCDTKY) MBROPT(*REPLACE)");
                            try
    {
      System.out.println("Se están copiando registros de QIWS/QCUSTCDT en "
+ library + "/QCUSTCDTKY...");
      c.run();
      AS400Message[] msgs = c.getMessageList();
      if (!msgs[0].getID().equals("CPC2955"))
      {
      System.out.println("No se ha podido llenar con datos " + library +
"/QCUSTCDTKY");
        for (int i = 0; i < msgs.length; i++)
        {
          System.out.println(msgs[i]);
        }
       System.exit(0);
      }
    }
    catch(Exception e)
    {
```

```
      System.out.println("No se ha podido llenar con datos " + library +
"/QCUSTCDTKY");
       System.exit(0);
    }
  }
}
```

Ejemplo: cómo se utiliza UserList para listar todos los usuarios de un grupo determinado

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de lista de usuarios. Este programa lista todos los usuarios de
un
// grupo determinado.
//
// Sintaxis de mandato:
//    UserListExample sistema grupo
//
// Este fuente es un ejemplo de "UserList" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import java.util.Enumeration;

public class UserListExample
{


    public static void main (String[] args)
    {
        // De no haberse especificado un sistema y un grupo, visualizar
        // texto de ayuda y salir.
        if (args.length != 2)
        {
            System.out.println("Utilización:  UserListExample sistema
grupo");
            return;
        }

                            try
        {
            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // El nombre de grupo se ha pasado como segundo argumento de la
            // línea de mandatos.
            String groupName = args[1];

            // Cree el objeto de lista de usuarios.
    UserList userList = new UserList (system);

            // Obtenga una lista de los usuarios del grupo especificado.
            userList.setUserInfo (UserList.MEMBER);
            userList.setGroupInfo (groupName);
    Enumeration enum = userList.getUsers ();

            // Itere por la lista e imprima los nombres y las descripciones
            // de los usuarios.
      while (enum.hasMoreElements())
```

```
            {
        User u = (User) enum.nextElement ();
                System.out.println ("Nombre de usuario:   " + u.getName ());
                System.out.println ("Descripción: " + u.getDescription ());
System.out.println("");            }

        }
        catch (Exception e)
        {
           System.out.println ("Error: " + e.getMessage ());
        }

        System.exit (0);
    }


}
```

Ejemplos: JavaBeans

En esta sección figura una lista de los ejemplos de código que se proporcionan en toda la documentación de los temas de beans.

- Ejemplo: utilizar escuchadores para imprimir un comentario al conectarse al sistema, desconectarse del mismo y ejecutar mandatos
- Ejemplo: utilizar applets e IBM VisualAge para Java para crear botones y ejecutar mandatos

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplo: código de bean de IBM Toolbox para Java

El siguiente ejemplo crea un objeto AS400 y un objeto CommandCall y luego registra escuchadores en los objetos. Los escuchadores de los objetos imprimen un comentario cuando el servidor se conecta o desconecta y cuando el objeto CommandCall completa la ejecución de un mandato.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de beans. Este programa utiliza el soporte de JavaBeans, en
// las clases de IBM Toolbox para Java.
//
// Sintaxis de mandato:
//    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()
    {
        // Siempre que el sistema se conecte o desconecte, imprima un
        // comentario. Para ello, añada un escuchador al objeto AS400.
        // Cuando un sistema se conecte o desconecte, el objeto AS400
        // llamará a este código.

        as400_.addConnectionListener
        (new ConnectionListener()
          {
             public void connected(ConnectionEvent event)
             {
                 System.out.println( "Sistema conectado." );
             }
             public void disconnected(ConnectionEvent event)
             {
                 System.out.println( "Sistema desconectado." );
             }
          }
        );

        // Siempre que un mandato se ejecute hasta completarse, imprima un
        // comentario. Para ello, añada un escuchador al objeto commandCall.
        // El objeto commandCall llamará a este código cuando ejecute un
mandato.

        cmd_.addActionCompletedListener(
            new ActionCompletedListener()
            {
```

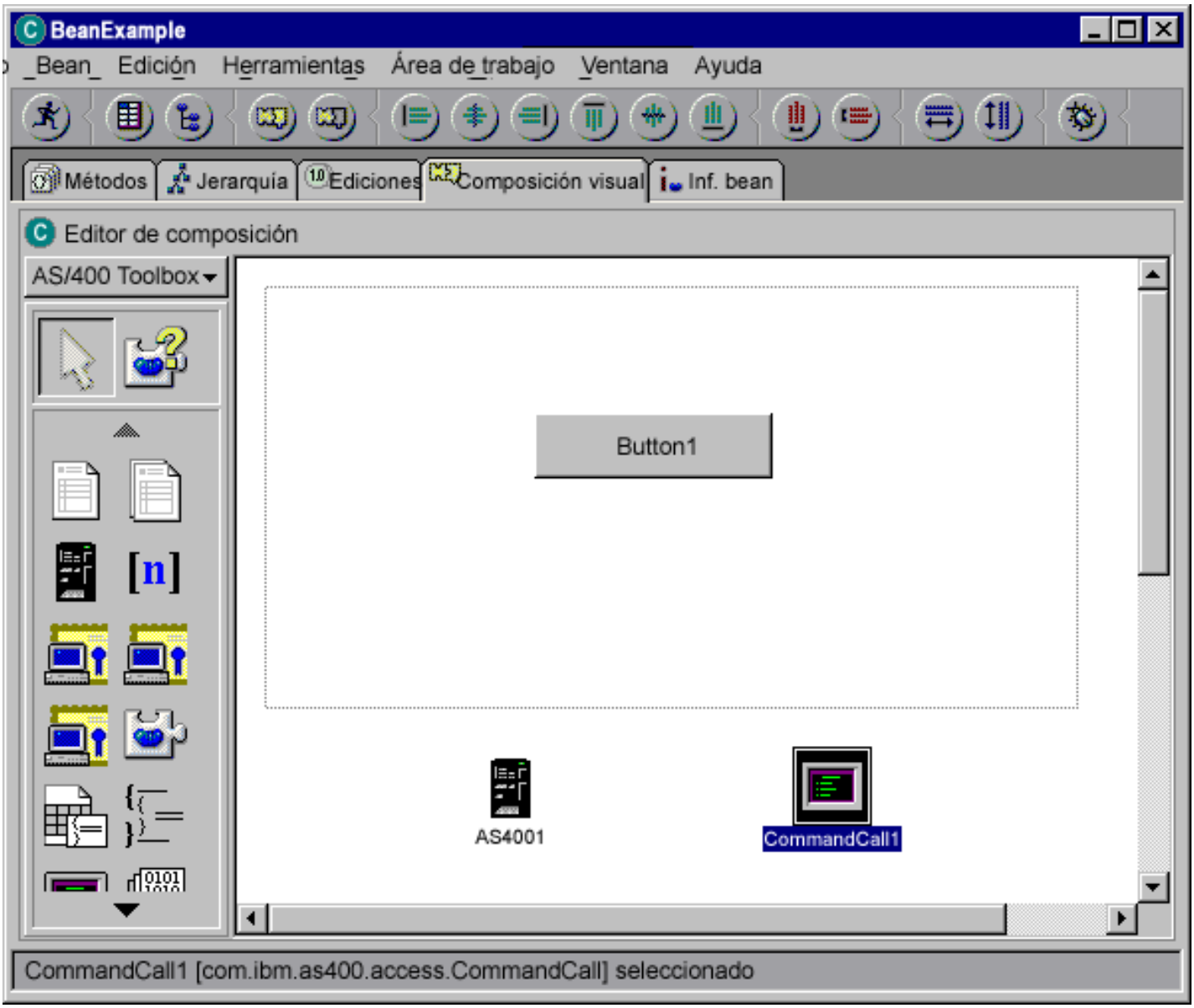
```
                public void actionCompleted(ActionCompletedEvent event)
                {
                    System.out.println( "Mandato completado." );
                }
            }
        );
    }

    void runCommand()
    {
                            try
        {
            // Ejecute un mandato. Los escuchadores imprimirán comentarios
            // al conectarse el sistema y cuando el mandato se haya
ejecutado
            // hasta completarse.
            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);
    }
}
```

Ejemplo de código de constructor visual de beans

En este ejemplo se utiliza el editor de composición de IBM VisualAge para Java, Enterprise Edition V2.0, pero los demás constructores visuales de beans son análogos. Este ejemplo crea un applet para un botón que, cuando se pulsa, ejecuta un mandato en el servidor iSeries o AS/400e.

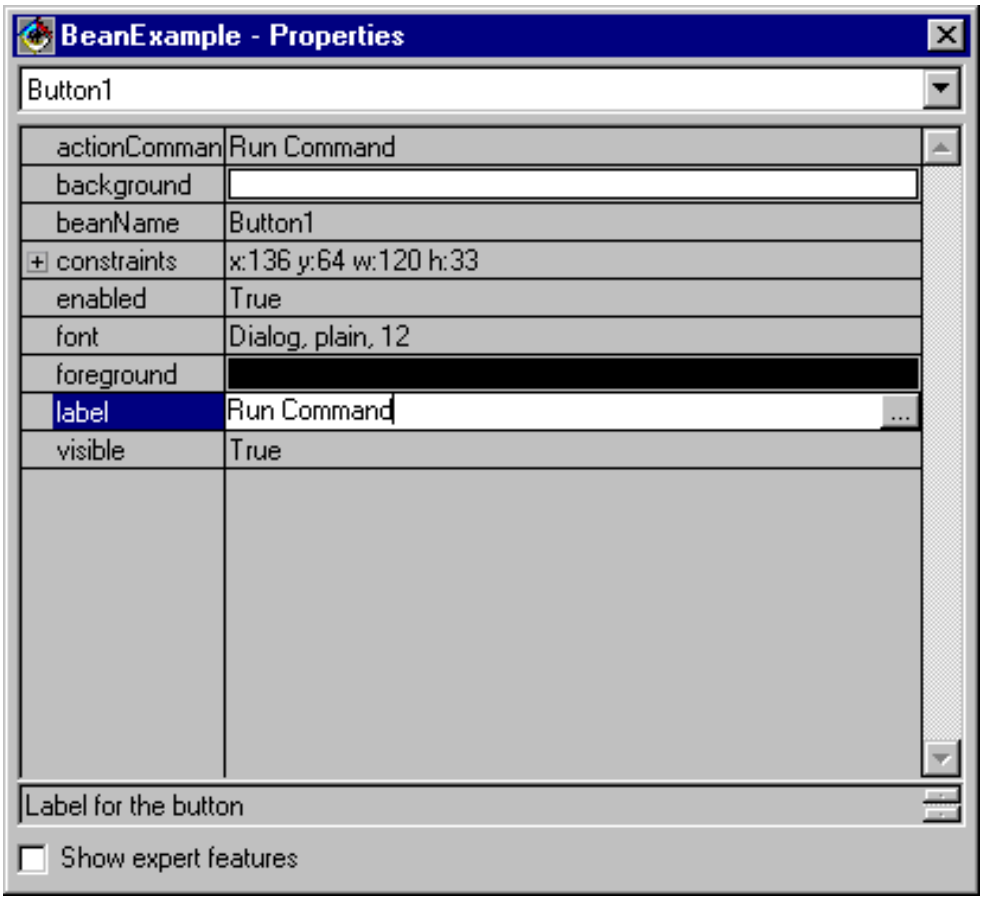
- Arrastre un botón y suéltelo en el applet. (El bean Button se halla en el constructor de beans, en la parte izquierda de la pestaña Composición visual, en la figura 1.)
- Suelte un bean CommandCall y un bean AS400 fuera del applet. (Los beans se hallan en el constructor de beans, en la parte izquierda de la pestaña Composición visual, en la figura 1.)

Figura 1: ventana Editor de composición visual de VisualAge - gui.BeanExample



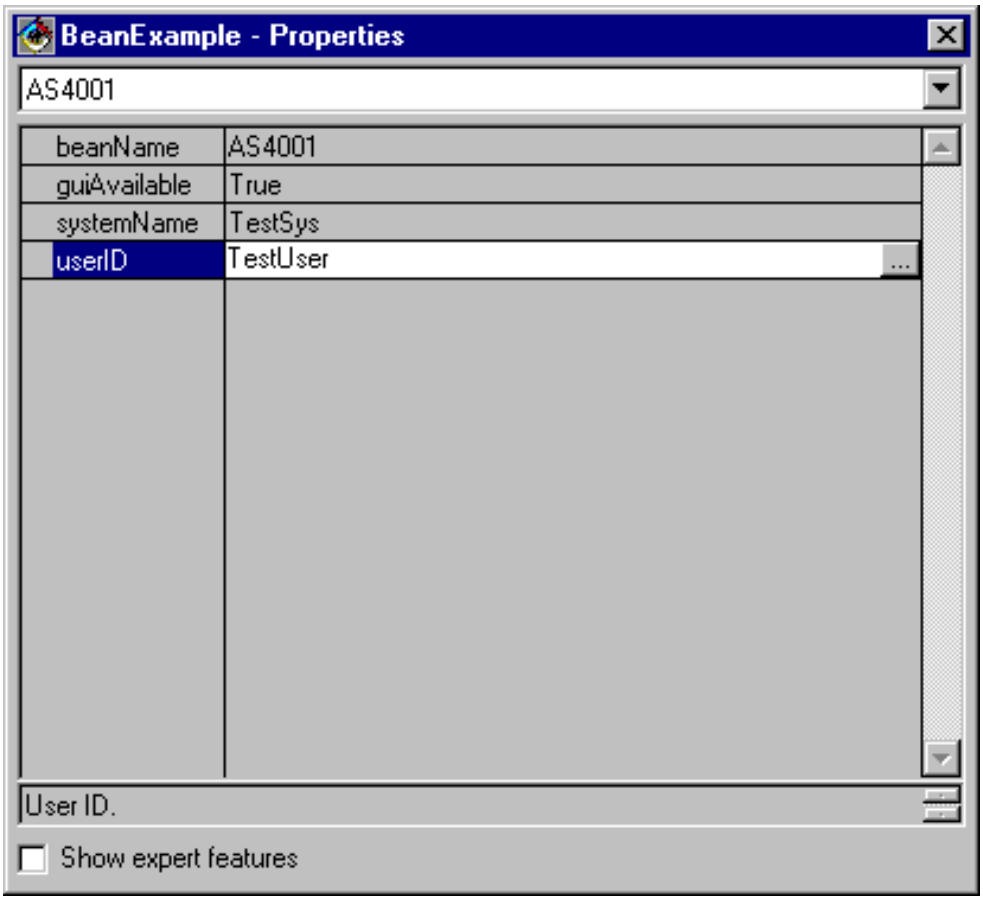
- Edite las propiedades del bean. (Para editarlas, seleccione el bean y luego pulse el botón derecho del ratón para visualizar una ventana emergente, una de cuyas opciones es Propiedades.)
  - Cambie la etiqueta del bean Button por **Ejecutar mandato**, como se muestra en la figura 2.

    Figura 2: cambiar la etiqueta del botón por Ejecutar mandato



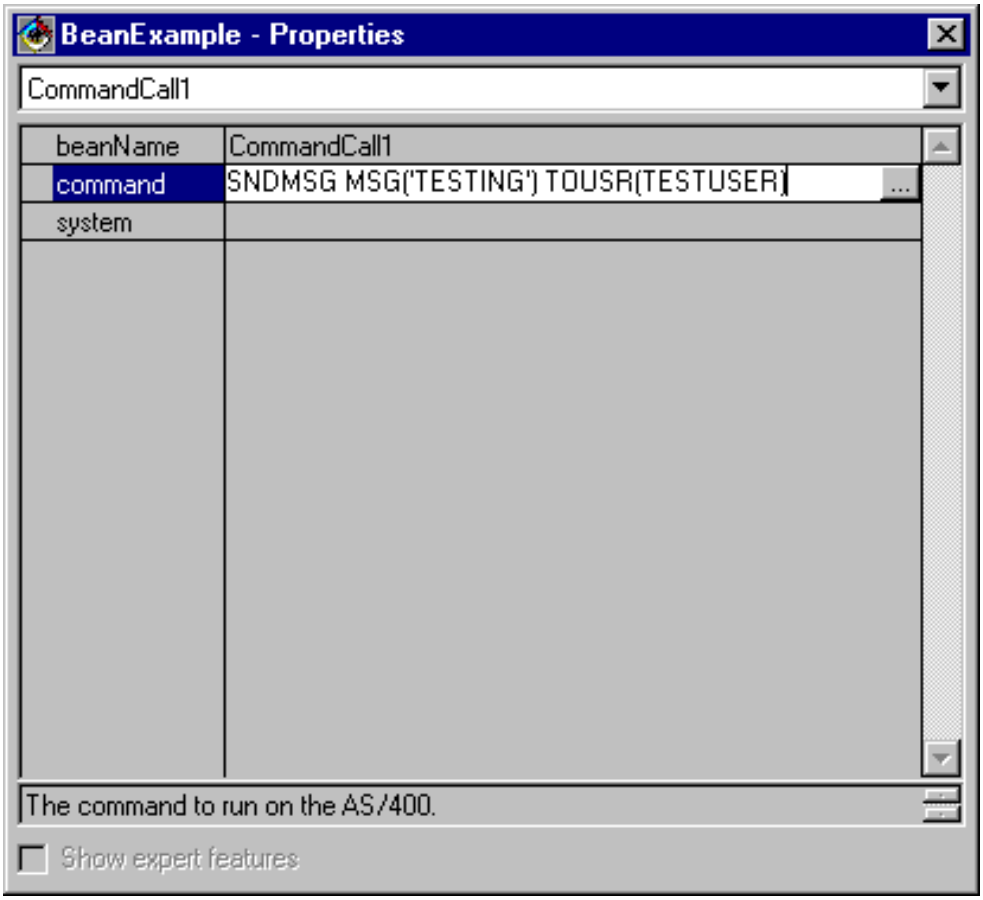
- Cambie el nombre de sistema del bean AS400 por **TestSys**.
- Cambie el ID de usuario del bean AS400 por **TestUser**, como se muestra en la figura 3.

Figura 3: cambiar el nombre del ID de usuario por TestUser



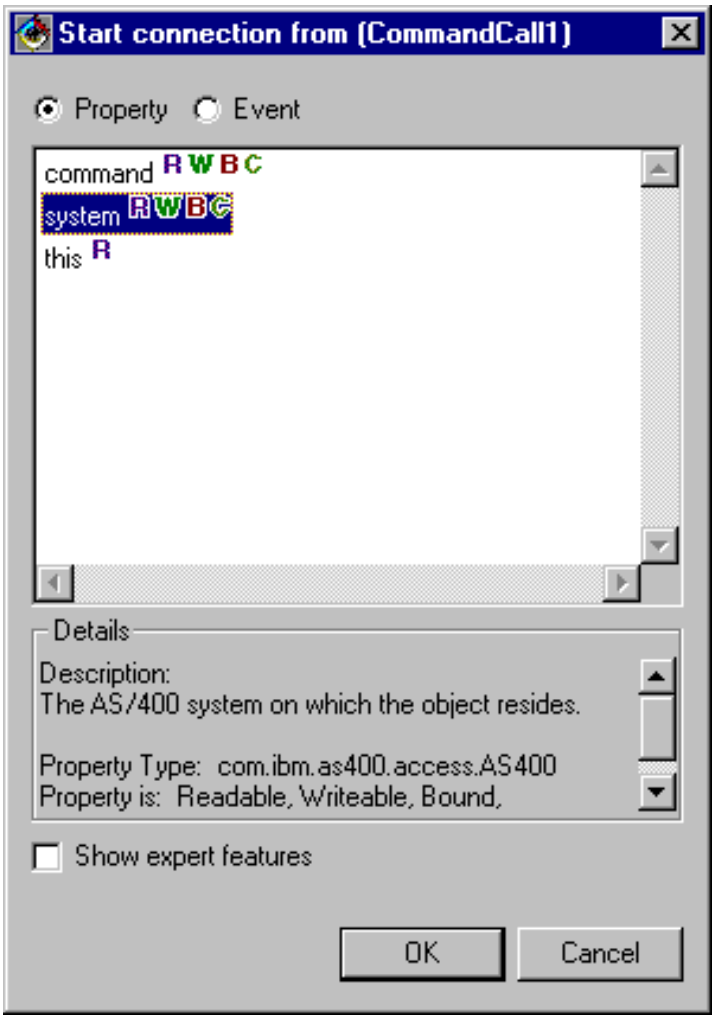
- Cambie el mandato del bean CommandCall por **SNDMSG MSG('Testing') TOUSR('TESTUSER')**, como se muestra en la figura 4.

Figura 4: cambiar el mandato del bean CommandCall



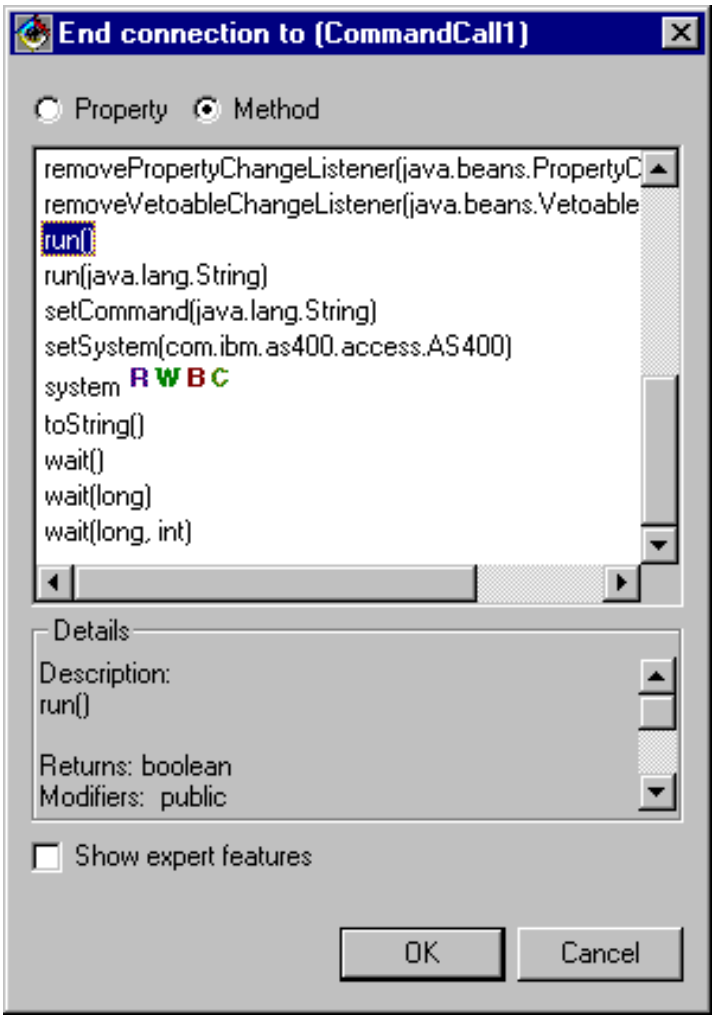
- Conecte el bean AS400 al bean CommandCall. El método que se utiliza para ello varía en función de los constructores de beans. Para este ejemplo, realice estos pasos:
    - Seleccione el bean CommandCall y luego pulse el botón derecho del ratón
    - Seleccione **Conectar**
    - Seleccione **Características conectables**
    - Seleccione **system** en la lista de características, como se muestra en la figura 5.
    - Seleccione el bean AS400
    - Seleccione **this** en el menú emergente que aparece sobre el bean AS400

Figura 5: conectar el bean AS400 al bean CommandCall



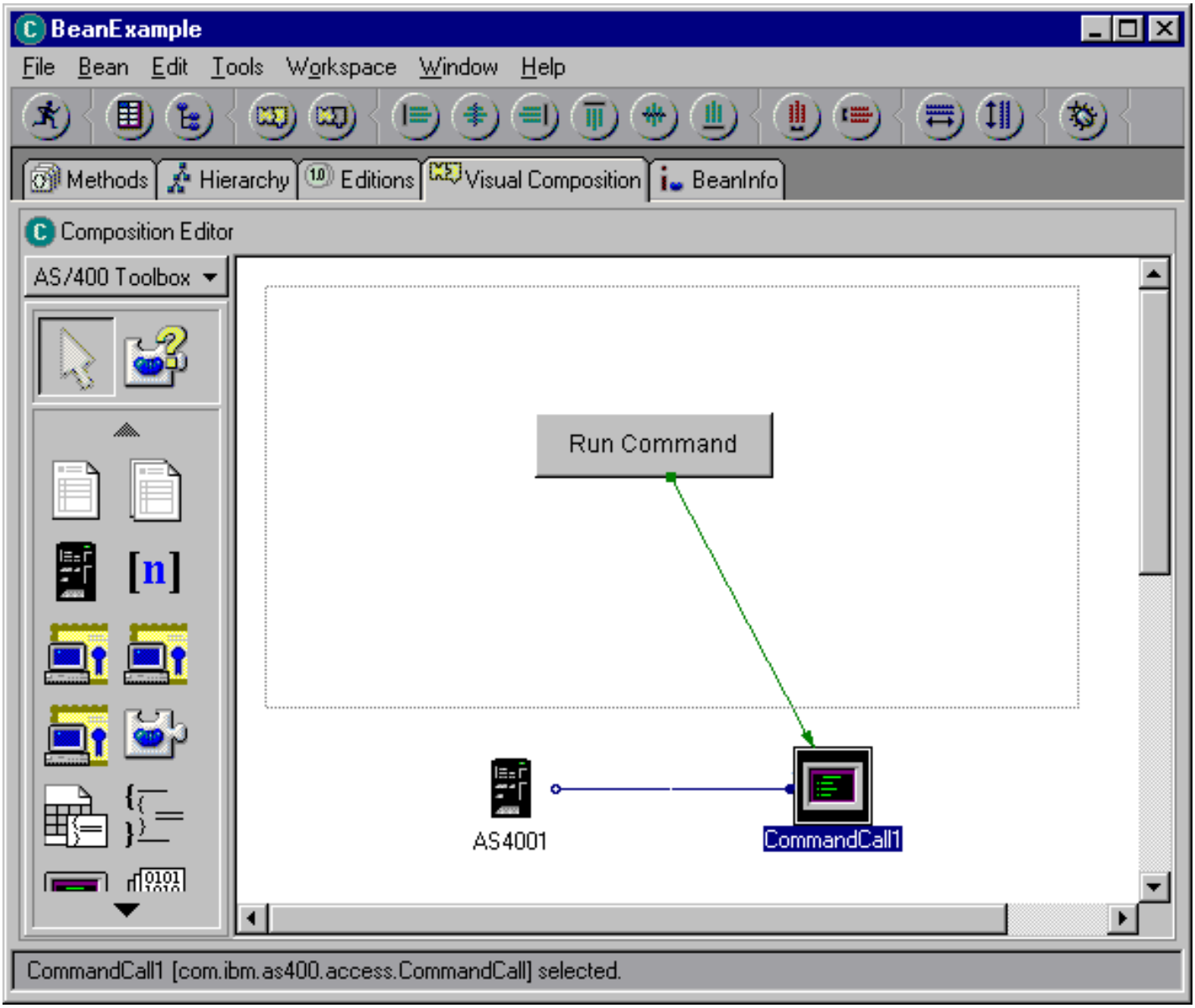
- Conecte el botón al bean CommandCall.
    - Seleccione el bean Button y luego pulse el botón derecho del ratón
    - Seleccione **Conectar**
    - Seleccione **actionPerformed**
    - Seleccione el bean CommandCall
    - Seleccione **Características conectables** en el menú emergente que aparece
    - Seleccione **run()** en la lista de métodos, como se muestra en la figura 6.

Figura 6: conectar un método a un botón



Cuando haya terminado, la ventana Editor de composición visual de VisualAge tendrá el aspecto de la figura 7.

Figura 7: ventana Editor de composición visual de VisualAge - Ejemplo de bean finalizado


BeanExample
File Bean Edit Tools Workspace Window Help
Methods Hierarchy Editions Visual Composition BeanInfo
Composition Editor
AS/400 Toolbox
Run Command
AS4001
CommandCall1
CommandCall1 [com.ibm.as400.access.CommandCall] selected.

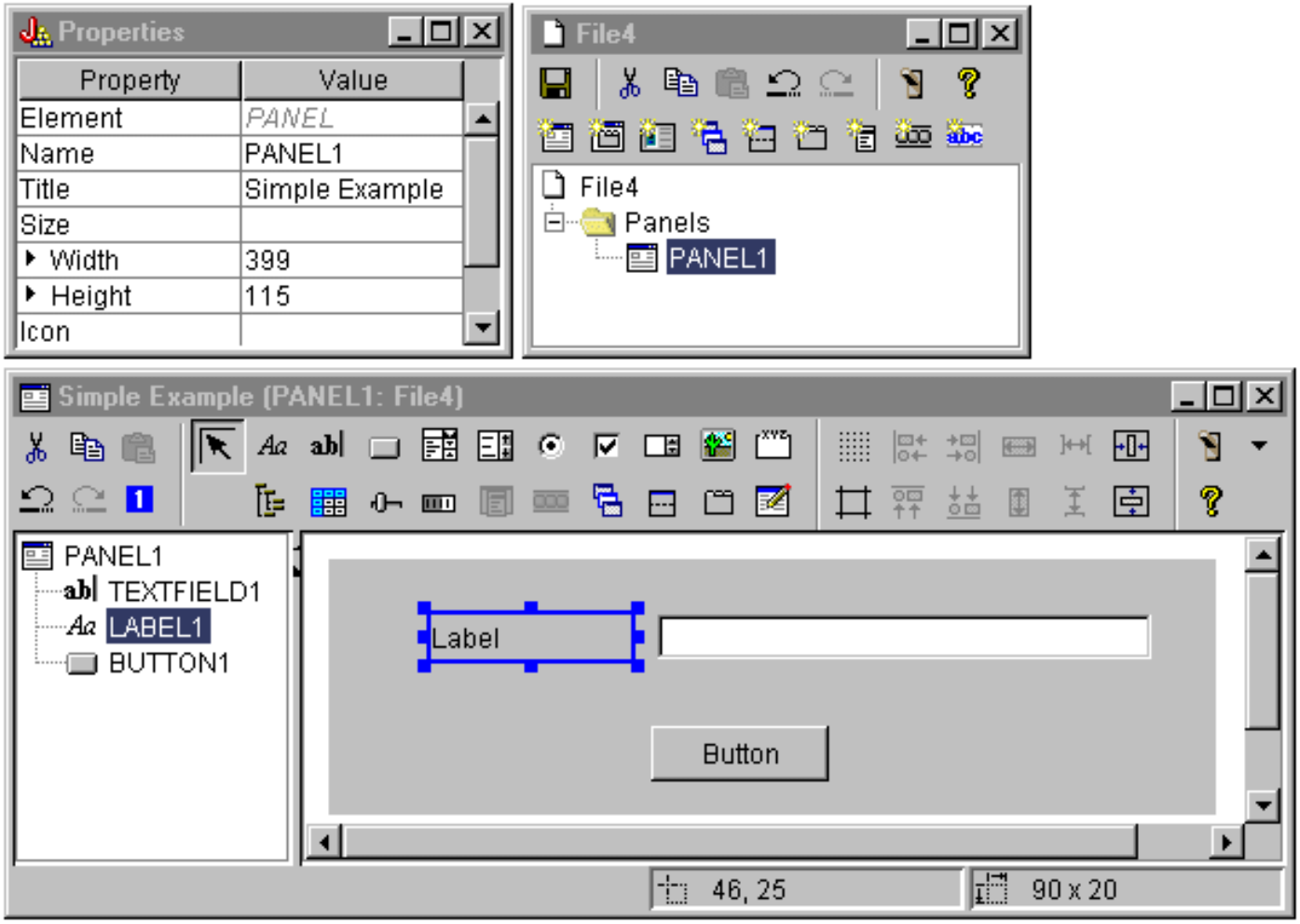
Ejemplo: construir un panel con el Constructor de GUI

Este ejemplo muestra cómo se utiliza la Caja de Herramientas Gráfica mediante la construcción de un simple panel. Es una visión general que ilustra las características y el funcionamiento básicos del entorno de la Caja de Herramientas Gráfica. Después de enseñarle cómo se construye un panel, el ejemplo le mostrará cómo se construye una pequeña aplicación Java que visualice el panel. En este panel de ejemplo, el usuario entra los datos en un campo de texto y pulsa el botón **Cerrar**. Luego la aplicación devuelve (se hace eco de) los datos a la consola Java.

Construcción del panel

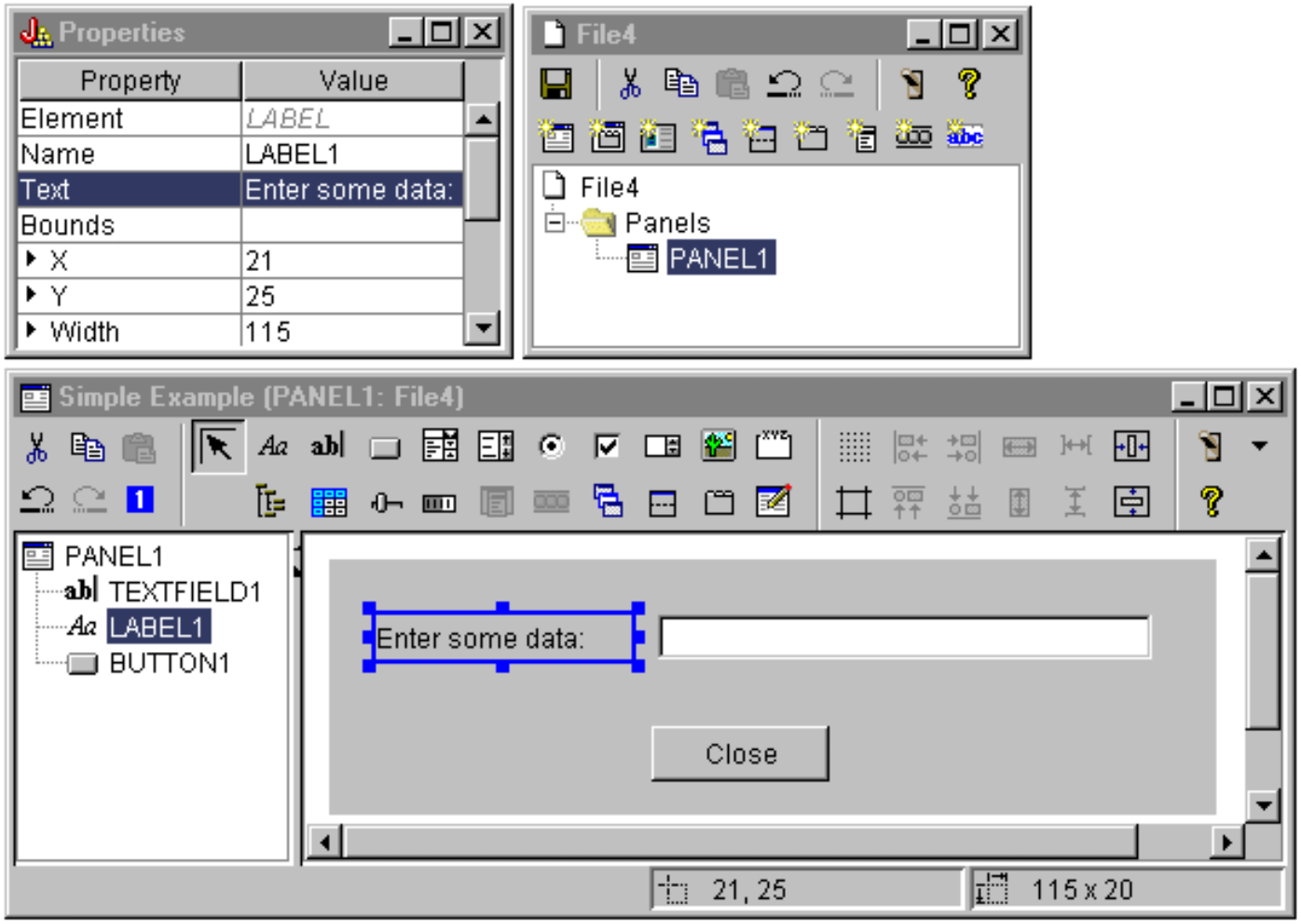
Al iniciar el Constructor de GUI, aparecen la ventana Propiedades y la ventana Constructor de GUI. Cree un archivo nuevo llamado "MyGUI.pdml". En el caso de este ejemplo, inserte un panel nuevo. Pulse el icono "Insertar panel" de la ventana Constructor de archivos. Su nombre es "PANEL1". Cambie el título del panel; para ello, modifique la información de la ventana Propiedades escribiendo "Ejemplo simple" en el campo "Título". Elimine los tres botones que aparecen por omisión seleccionándolos con el ratón y pulsando "Suprimir". Con los botones que hay en la ventana Constructor de paneles, añada los tres elementos que se indican en la figura 1: una etiqueta, un campo de texto y un pulsador.

Figura 1: ventanas del Constructor de GUI - Empezar a construir un panel



Si selecciona la etiqueta, podrá cambiar su texto en la ventana Propiedades. En este ejemplo, se ha llevado a cabo la misma acción para el pulsador, cambiando el texto por "Cerrar".

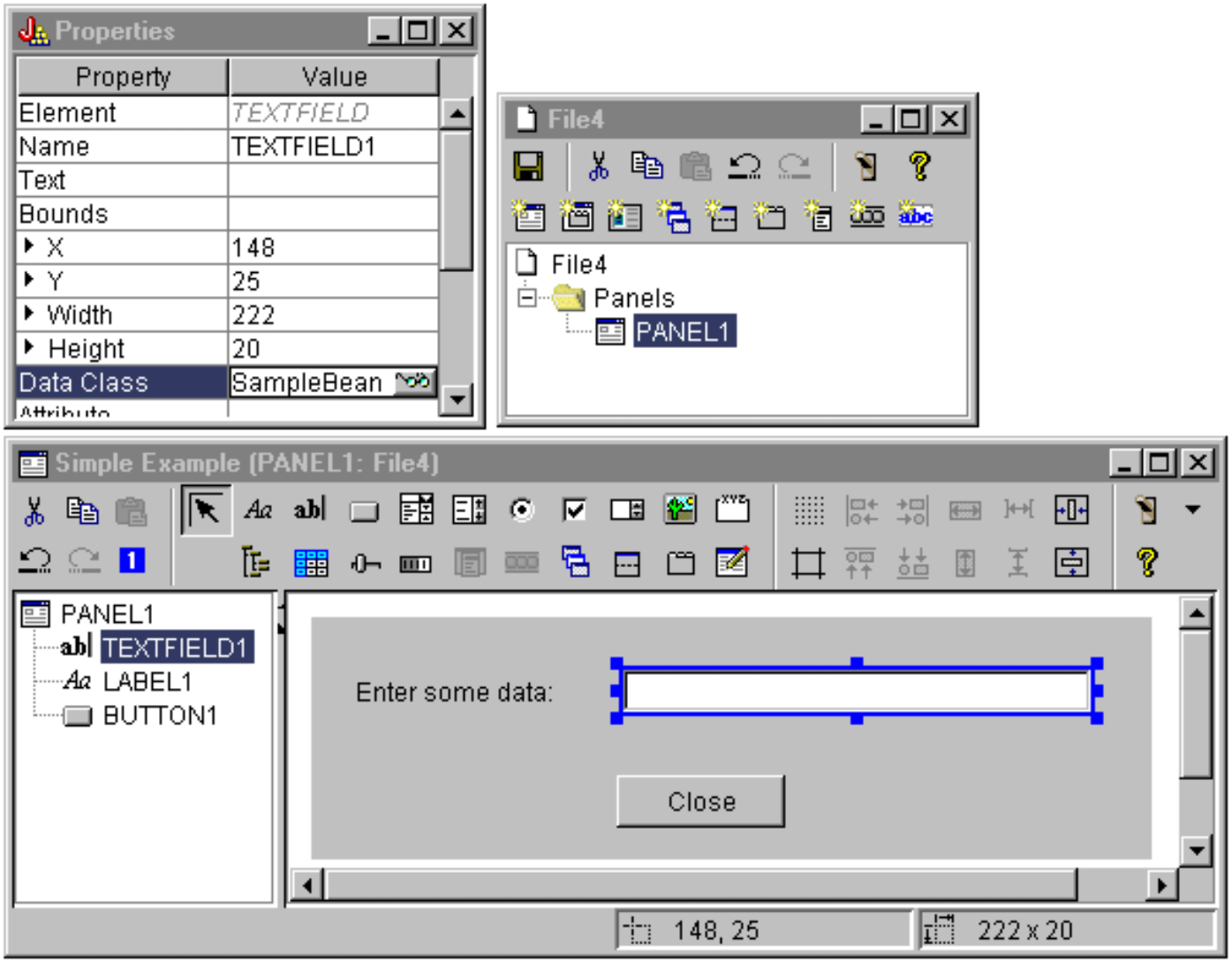
Figura 2: ventanas del Constructor de GUI - Cambiar el texto en la ventana Propiedades



Campo de texto

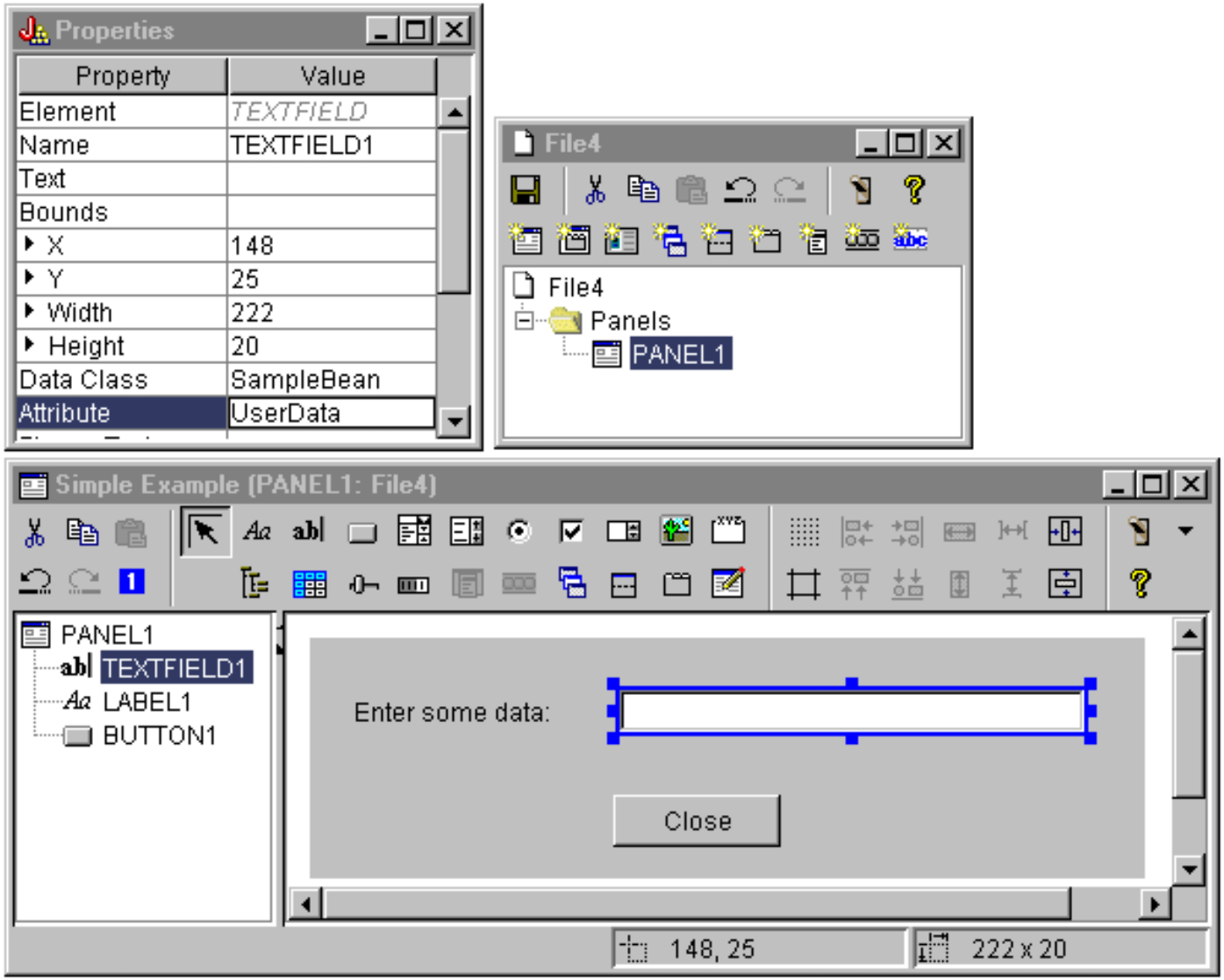
El campo de texto contendrá datos y, por lo tanto, se podrán establecer varias propiedades que permitan al Constructor de GUI hacer algunas tareas adicionales. En este ejemplo, la propiedad DataClass se establece en el nombre de una clase de bean llamada **SampleBean**. Este bean de datos suministrará los datos a este campo de texto.

Figura 3: ventanas del Constructor de GUI - Establecer la propiedad DataClass



Establezca la propiedad atributo (Attribute) en el nombre de la propiedad de bean que va a contener los datos. En este caso, el nombre es **UserData**.

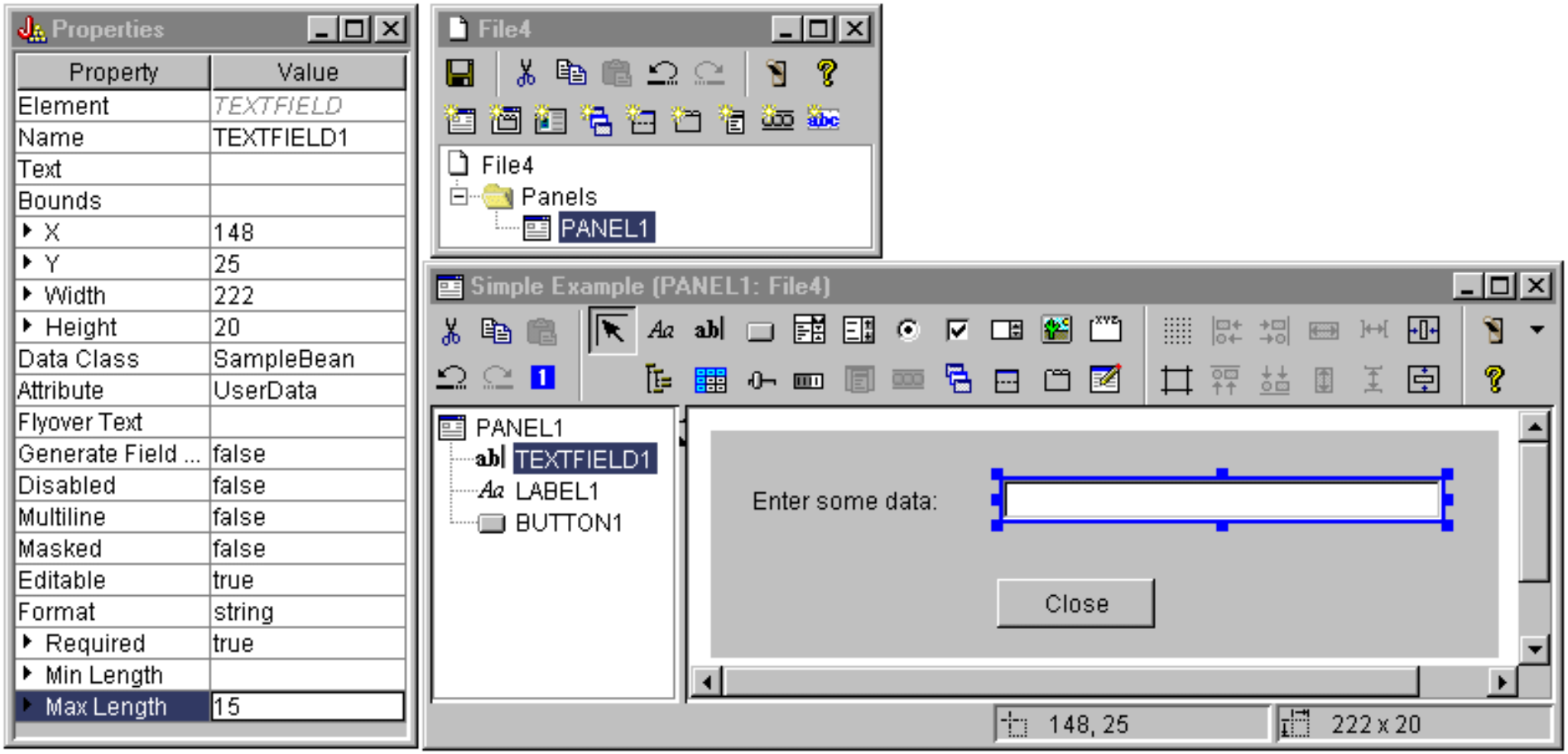
Figura 4: ventanas del Constructor de GUI - Establecer la propiedad Attribute



Si se siguen los pasos anteriores, la propiedad **UserData** quedará enlazada a este campo de texto. La Caja de Herramientas Gráfica, para obtener en tiempo de ejecución el valor inicial de este campo, llama a **`SampleBean.getUserData`**. Cuando el panel se cierra, para enviar de regreso a la aplicación el valor modificado, se llama a **`SampleBean.setUserData`**.

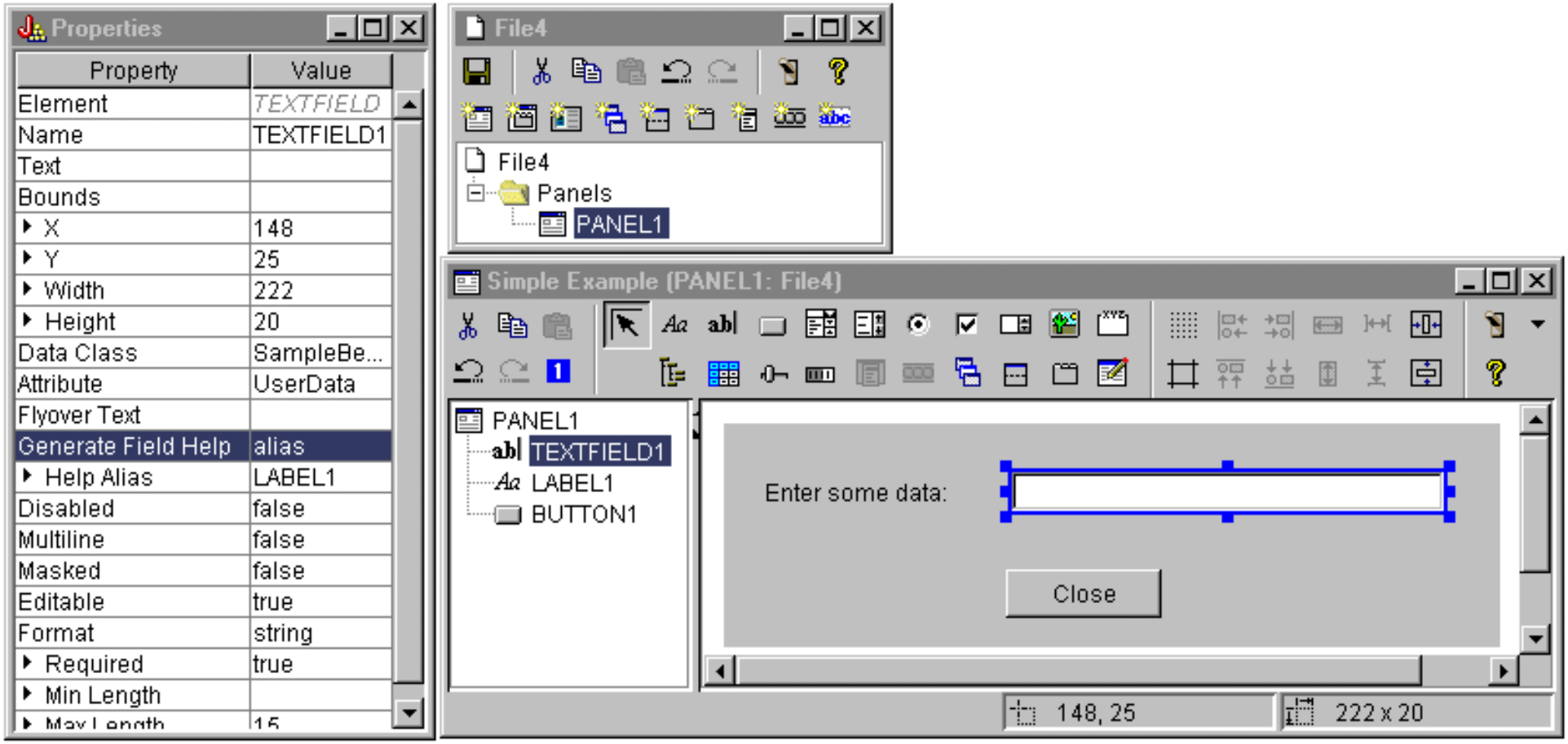
Especifique que al usuario se le exigirán algunos datos y que esos datos deben ser una serie de como máximo 15 caracteres.

Figura 5: ventanas del Constructor de GUI - Establecer la longitud máxima del campo de texto



Indique que la ayuda según contexto del campo de texto va a ser el tema de ayuda asociado a la etiqueta para especificar algunos datos.

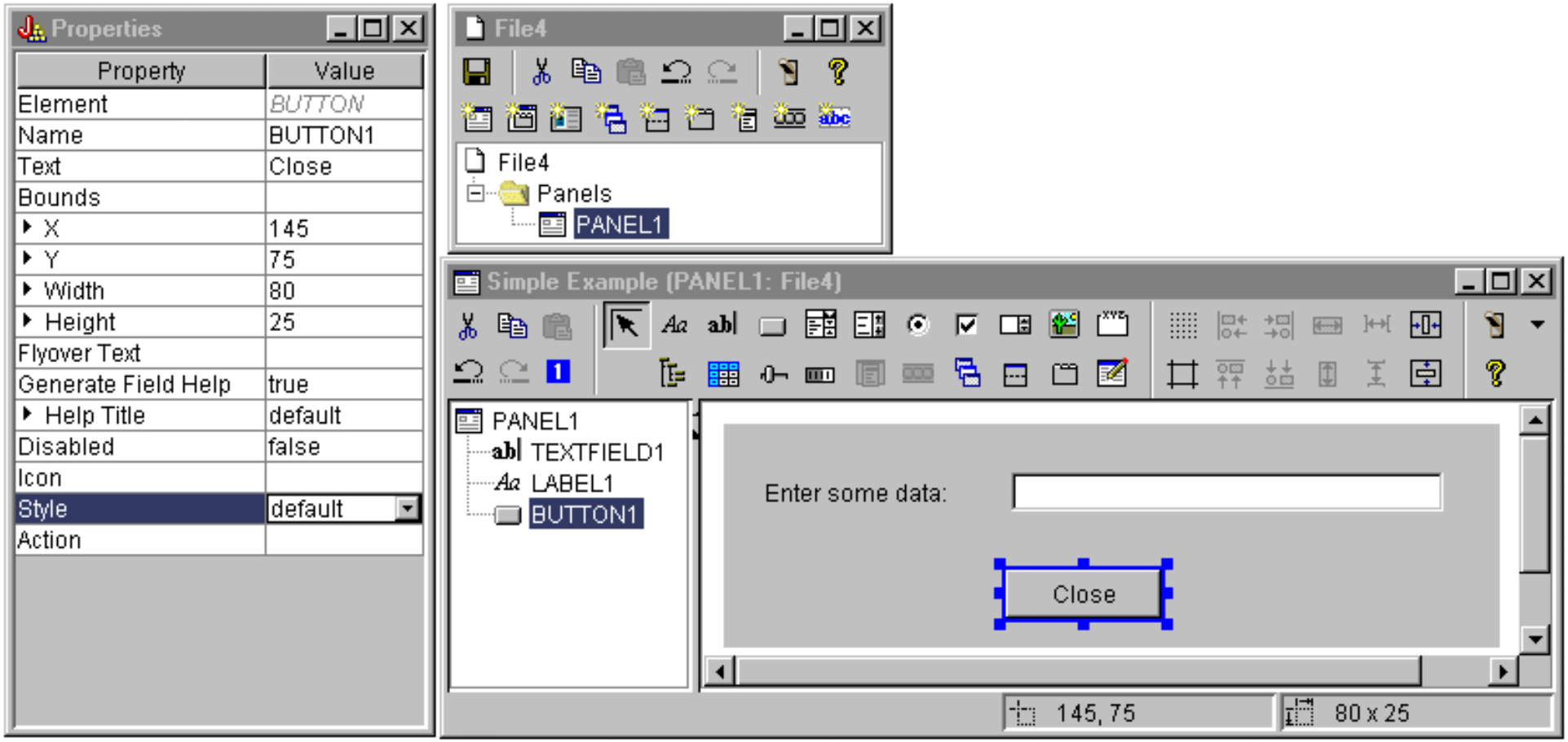
Figura 6: ventanas del Constructor de GUI - Establecer la ayuda según contexto para el campo de texto



Botón

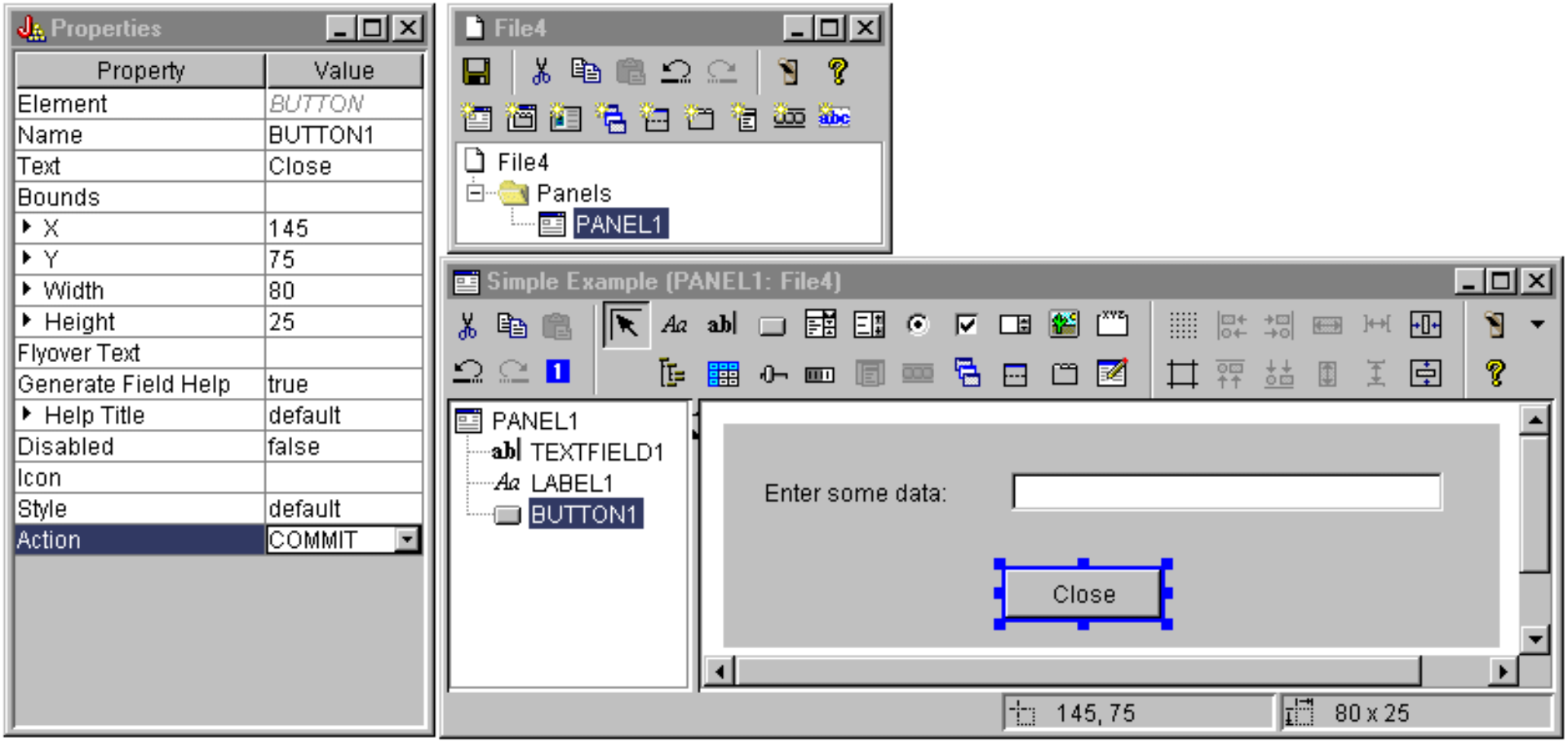
Modifique la propiedad style para dar al botón el énfasis por omisión.

Figura 7: ventanas del Constructor de GUI - Establecer la propiedad Style para dar al botón el énfasis por omisión



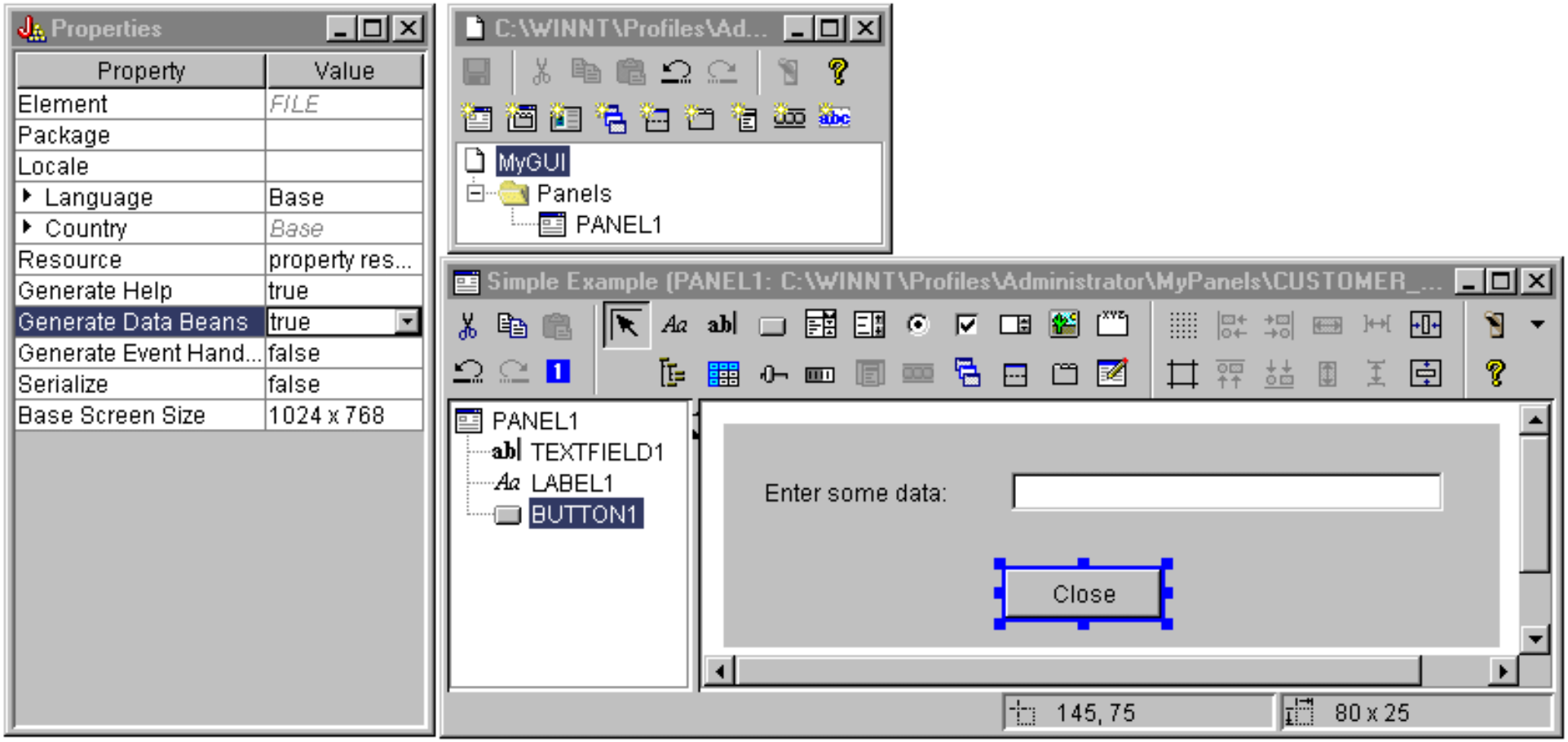
Establezca la propiedad ACTION en COMMIT; esto hace que se llame al método `setUserData` del bean cuando se seleccione el botón.

Figura 8: ventanas del Constructor de GUI - Establecer la propiedad Action en COMMIT



Antes de guardar el panel, establezca propiedades al nivel del archivo PDML para generar tanto el esqueleto de la ayuda en línea como el bean Java. A continuación, para guardar el archivo, pulse el icono de la ventana del Constructor de GUI. Cuando se lo soliciten, especifique el nombre de archivo **MyGUI.pdml**.

Figura 9: ventanas del Constructor de GUI - Establecer propiedades para generar el esqueleto de la ayuda en línea y el bean Java



Archivos generados

Después de guardar la definición de panel, puede echar una ojeada a los archivos generados por el Constructor de GUI. **Archivo PDML** Aquí está el contenido de **MyGUI.pdml** para darle una idea de cómo funciona el lenguaje PDML (Panel Definition Markup Language). Debido a que el PDML sólo se utiliza mediante las herramientas proporcionadas por la Caja de Herramientas Gráfica, no es necesario que comprenda con todo detalle el formato de este archivo:

```
<!-- Generado por el Constructor de GUI -->
<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>
```

Paquete de recursos

Asociado a todo archivo PDML hay un paquete de recursos. En este ejemplo, los recursos traducibles se han guardado en un archivo PROPERTIES llamado **MyGUI.properties**. Fíjese que el archivo PROPERTIES también contiene datos de personalización para el Constructor de GUI.

```
##Generado por el Constructor de GUI
BUTTON_1=Cerrar
TEXT_1=
@GenerateHelp=1
@Serialize=0
@GenerateBeans=1
LABEL_1=Entre algunos datos:
PANEL_1.Margins=18,18,18,18,18,18
PANEL_1=Ejemplo simple
```

JavaBean

El ejemplo también ha generado un esqueleto de código fuente Java para el objeto JavaBean. A continuación figura el contenido de **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 = "";
    }
}
```

Observe que el esqueleto ya contiene una implementación de los métodos obtenedores y establecedores para la propiedad `UserData`. Los demás métodos se definen mediante la interfaz `DataBean` y, por consiguiente, son obligatorios.

El Constructor de GUI ya ha invocado el compilador Java para obtener el esqueleto y ha generado el correspondiente archivo de clase. En este sencillo ejemplo, no es necesario modificar la implementación del bean. En una aplicación Java real, normalmente se tendrían que modificar los métodos `load` y `save` para transferir los datos desde un origen de datos externo. A menudo es suficiente con la implementación por omisión de los otros dos métodos. Encontrará más información al respecto en la documentación sobre la interfaz **`DataBean`** en los javadocs de la infraestructura en tiempo de ejecución PDML.

Archivo de ayuda

El Constructor de GUI también crea una infraestructura HTML llamada documento de ayuda. A los transcriptores de ayuda les resulta muy fácil gestionar la información de ayuda mediante la edición de este archivo. Encontrará más información en estos temas:

- Creación del documento de ayuda
- Edición de los documentos de ayuda generados por el Constructor de GUI

Construcción de la aplicación

Tras guardar la definición de panel y los archivos generados, ya está usted listo para construir la aplicación. Todo lo que necesita es un nuevo archivo fuente Java que contenga el punto de entrada principal para la aplicación. En este ejemplo, el archivo se llama **SampleApplication.java**. Contiene el código siguiente:

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

public class SampleApplication
{
    public static void main(String[] args)
    {
        // Cree una instancia del objeto bean que suministra datos al panel
        SampleBean bean = new SampleBean();

        // Inicialice el objeto
        bean.load();

        // Configure que se pase el bean al gestor de paneles
        DataBean[] beans = { bean };

       // Cree el gestor de paneles.Parámetros:
        // 1. Archivo PDML como nombre de recurso
        // 2. Nombre del panel que se ha de visualizar
        // 3. Lista de objetos de datos que suministran los datos del panel
        // 4. Un marco AWT para que el panel sea modal

        PanelManager pm = null;
        try { pm = new PanelManager("MyGUI", "PANEL_1", beans, new Frame()); }
        catch (DisplayManagerException e)
        {
            // Algo no ha funcionado; se ha de visualizar un mensaje y salir
            e.displayUserMessage(null);
            System.exit(1);
        }

        // Muestre el panel - aquí se cede el control
       pm.setVisible(true);

        // Devuelva los datos de usuario guardados
        System.out.println("DATOS DE USUARIO GUARDADOS: '" + bean.getUserData() + "'");

        // Salga de la aplicación
        System.exit(0);
    }
}
```

El programa llamador es el encargado de inicializar los objetos bean (uno o varios) mediante una llamada a **`load`**. Si son varios los objetos bean que suministran los datos de un panel, cada uno de ellos debe inicializarse antes de que se pasen al entorno de la Caja de Herramientas Gráfica.

La clase **`com.ibm.as400.ui.framework.java.PanelManager`** suministra la API que permite visualizar ventanas y diálogos autónomos. La Caja de Herramientas Gráfica trata como nombre de recurso el nombre del archivo PDML tal como se suministra en el constructor. En la vía de acceso de clases (classpath) debe estar identificado el directorio, el archivo ZIP o el archivo JAR que contiene el PDML.

Debido a que en el constructor se suministra un objeto marco (**`Frame`**), la ventana se comportará como diálogo modal. En una aplicación Java real, este objeto podría obtenerse de una

ventana padre adecuada para el diálogo. Dado que la ventana es modal, el control no vuelve a la aplicación hasta que el usuario cierra la ventana. Llegado este momento, la aplicación no hace otra cosa que devolver (como un eco) los datos de usuario modificados y luego se cierra.

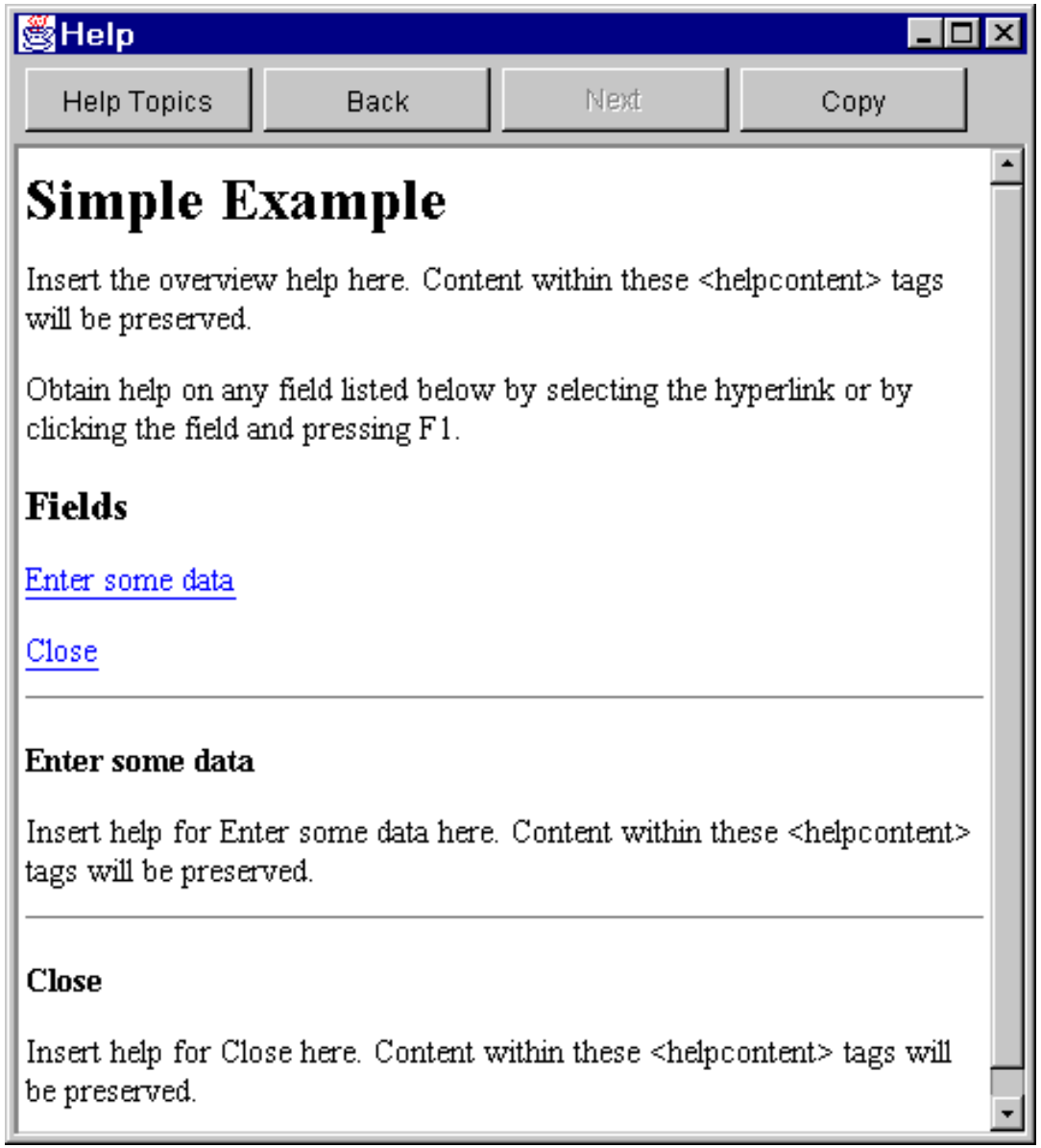
Ejecución de la aplicación

En esta figura puede ver el aspecto que ofrece la ventana, una vez compilada y ejecutada la aplicación:

Figura 10: ventana de la aplicación del ejemplo simple

Si el usuario pulsa la tecla F1 mientras el foco se encuentra en el campo de texto, la Caja de Herramientas Gráfica visualizará un navegador de ayuda que contiene el esqueleto de la ayuda en línea generado por el Constructor de GUI.

Figura 11: esqueleto de ayuda en línea del ejemplo simple



Puede editar el HTML y añadir contenido de ayuda real para los temas de ayuda mostrados.

Si los datos del campo de texto no son válidos (por ejemplo, si el usuario ha pulsado el botón **Cerrar** sin haber suministrado ningún valor), la Caja de Herramientas Gráfica visualiza un mensaje de error y devuelve el foco al campo para que puedan entrarse los datos.

Figura 12: Mensaje de error de datos

Encontrará más información sobre cómo se ejecuta este ejemplo como un applet en Utilización de la Caja de Herramientas Gráfica en un navegador.

Cuadros combinados editables

El generador de beans, cuando crea un método de obtención y un método de establecimiento para un cuadro combinado editable, devuelve por omisión un objeto String en el método de establecimiento y toma por omisión un parámetro de tipo serie en el método de obtención. Puede ser conveniente cambiar este comportamiento por omisión de tal forma que el método de establecimiento tome una clase Object y el método de obtención devuelva un tipo Object. Ello le permitiría determinar la selección de usuario mediante descriptores de elecciones (ChoiceDescriptor).

Si se detecta el tipo Object para el método de obtención y el de establecimiento, el sistema esperará un ChoiceDescriptor o un tipo Object, en vez de una serie formateada.

Ejemplo

Supongamos que Editable sea un cuadro combinado (ComboBox) editable que tenga un valor Double, que utilice un valor del sistema o bien que no esté establecido.

```
public Object getEditable()
{
    if (m_setting == SYSTEMVALUE)
    {
       return new ChoiceDescriptor("choice1","Valor del sistema");
    }
    else if (m_setting == NOTSET)
    {
       return new ChoiceDescriptor("choice2","Valor no establecido");
    }
          else
    {
       return m_doubleValue;
    }
}
```

De modo parecido, cuando se detecta el tipo Object para el método de obtención y el de establecimiento, el sistema devolverá un Object que sea un ChoiceDescriptor con la elección seleccionada o un tipo 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
    { /* proceso con error */ }
}
```

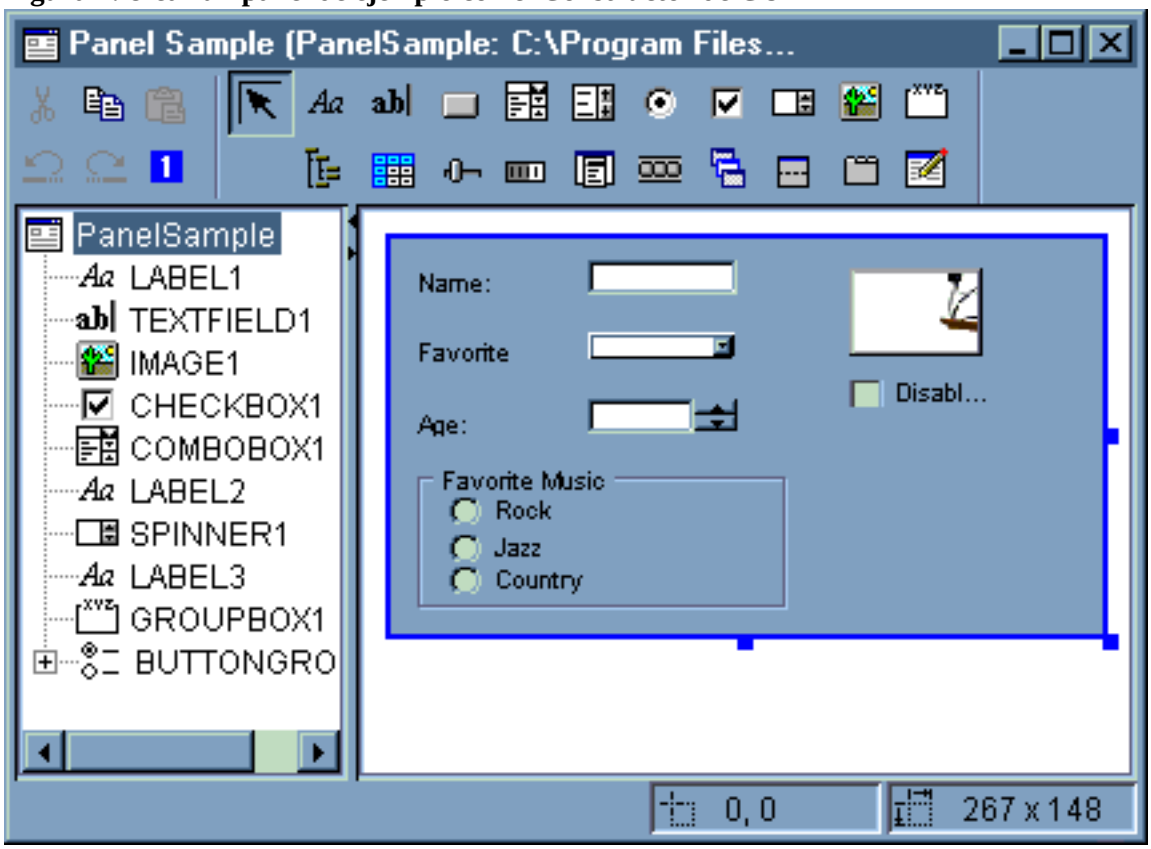
Crear un panel con el Constructor de GUI

Crear un panel con el Constructor de GUI resulta sencillo. En la barra de menús de la ventana principal del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

En la barra de menús de la ventana **Archivo** del Constructor de GUI, pulse el icono de Insertar panel nuevo  para visualizar un constructor de paneles donde puede insertar los componentes para el panel. Los botones de la barra de herramientas de la ventana **Panel** representan los diversos componentes que puede añadir al panel. Seleccione el componente que desee y después pulse en el lugar donde haya pensado situarlo.

La siguiente imagen muestra un panel que se ha creado con varias de las opciones disponibles.

Figura 1: crear un panel de ejemplo con el Constructor de GUI



El panel de ejemplo de la figura 1 utiliza el siguiente código de DataBean para reunir los diversos componentes:

```
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("Nombre = " + m_sName);
    System.out.println("Alimentos favoritos = " + m_oFavoriteFood);
    System.out.println("Edad = " + 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("Música favorita = " + sMusic);
    }

    public void load()
    {
        m_sName = "Nombre de ejemplo";
        m_oFavoriteFood = null;
        m_cdFavoriteFood = new ChoiceDescriptor[0];
        m_oAge = new Integer(50);
        m_sFavoriteMusic = "RADIOBUTTON1";
    }
}
```

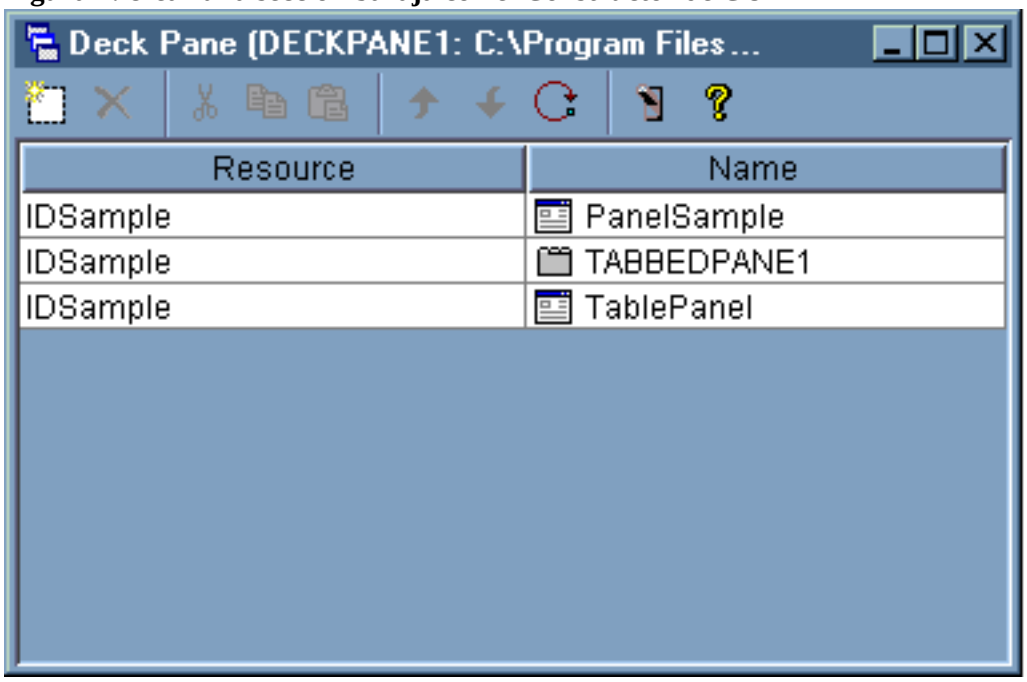
El panel es el componente más simple de que dispone el Constructor de GUI, pero a partir de él se pueden construir magníficas aplicaciones de interfaz de usuario (UI).

Crear una sección baraja con el Constructor de GUI

Con el Constructor de GUI es sencillo crear una sección baraja. En la barra de menús de la ventana del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

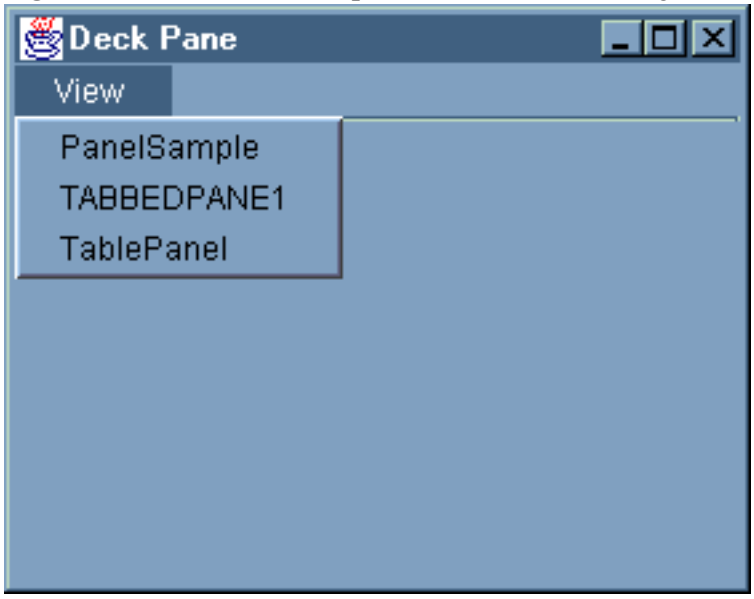
En la barra de menús de la ventana **Archivo** del Constructor de GUI, pulse el botón de la herramienta **Insertar sección baraja** para visualizar un constructor de paneles donde puede insertar los componentes para la sección baraja. En el ejemplo siguiente, se añaden tres componentes.

Figura 1: crear una sección baraja con el Constructor de GUI



Una vez creada la sección baraja, pulse el botón de la herramienta **Vista previa** para obtener una vista previa de la misma. Las secciones baraja aparecen como si fueran planas hasta que se selecciona el menú **Ver**.

Figura 1: obtener una vista previa de la sección baraja con el Constructor de GUI



En el menú **Ver**, seleccione el componente que desea ver. Para este ejemplo, puede elegir ver el ejemplo de panel (PanelSample), TABBEDPANE1, o el panel de tabla (TablePanel). Las figuras siguientes ilustran lo que se muestra al

ver estos componentes.

Figura 3: ver el ejemplo de panel (PanelSample) con el Constructor de GUI

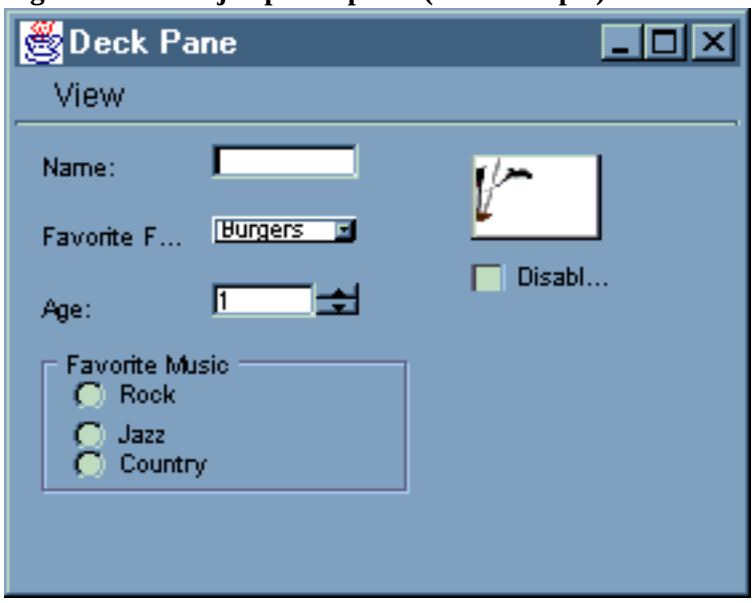


Figura 4: ver TABBEDPANE1 con el Constructor de GUI

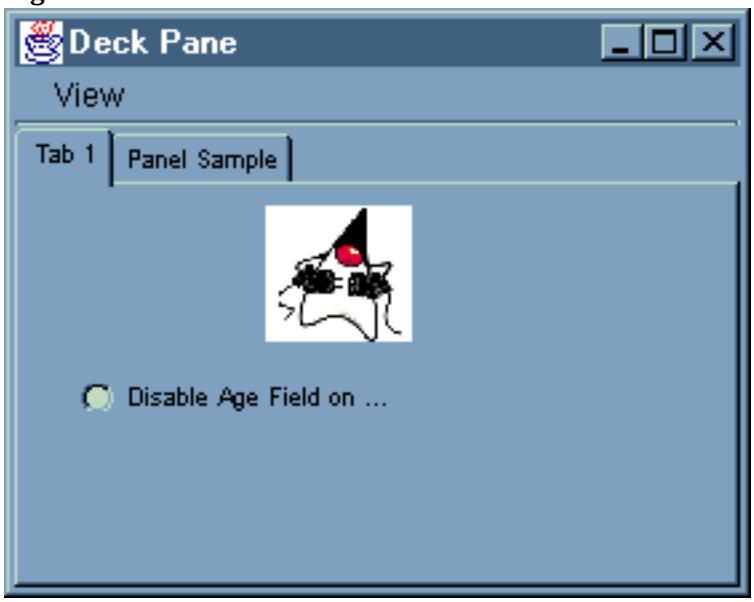
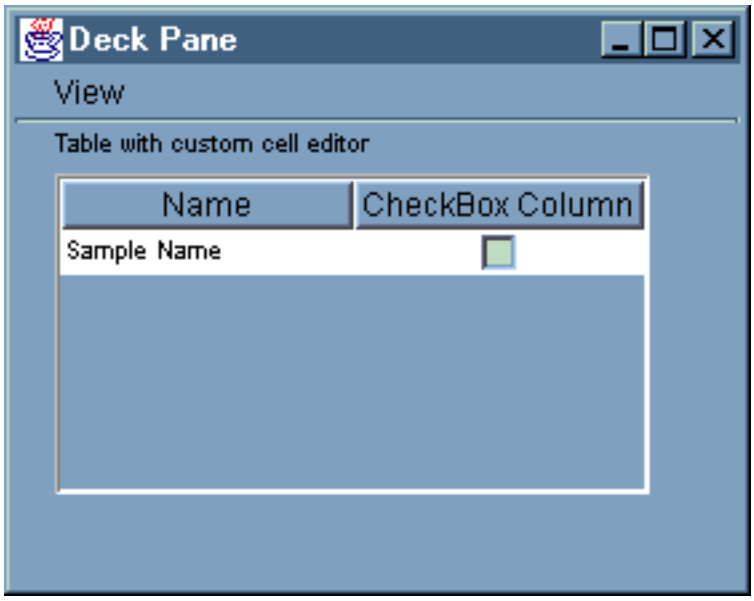


Figura 5: ver el panel de tabla (TablePanel) con el Constructor de GUI

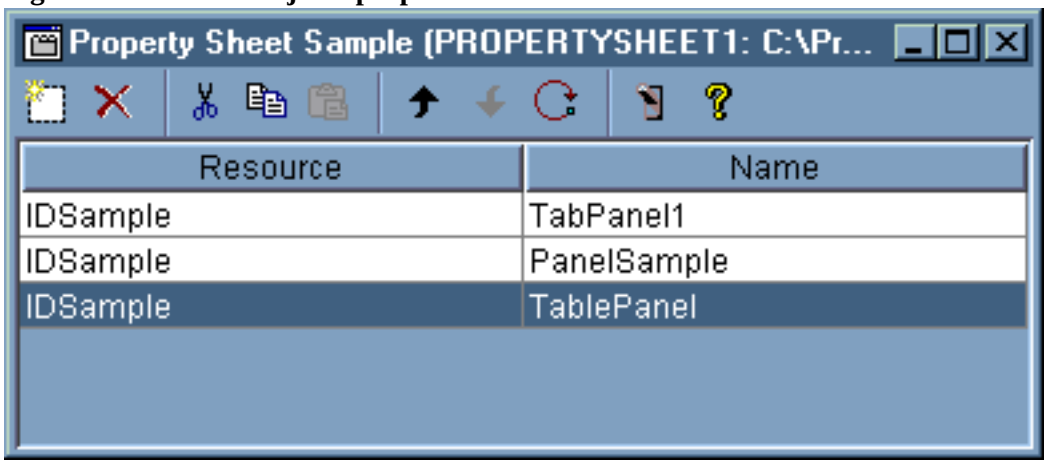

Deck Pane
View
Table with custom cell editor
Name
CheckBox Column
Sample Name

Crear una hoja de propiedades con el Constructor de GUI

Con el Constructor de GUI resulta muy simple crear una hoja de propiedades. En la barra de menús de la ventana principal del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

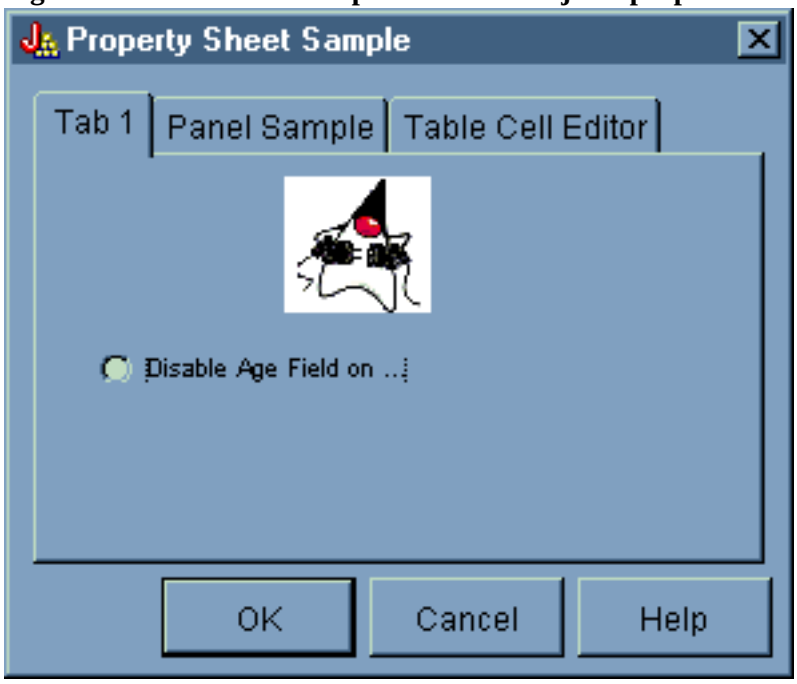
En la barra de menús de la ventana **Archivo** del Constructor de GUI, pulse el icono de Insertar hoja de propiedades  para visualizar un constructor de paneles donde puede insertar los componentes para la hoja de propiedades.

Figura 1: crear una hoja de propiedades con el Constructor de GUI



Cuando haya creado la hoja de propiedades, utilice el icono  para obtener una vista previa de la misma. En este ejemplo hay tres pestañas entre las que puede elegir.

Figura 2: obtener una vista previa de una hoja de propiedades con el Constructor de GUI

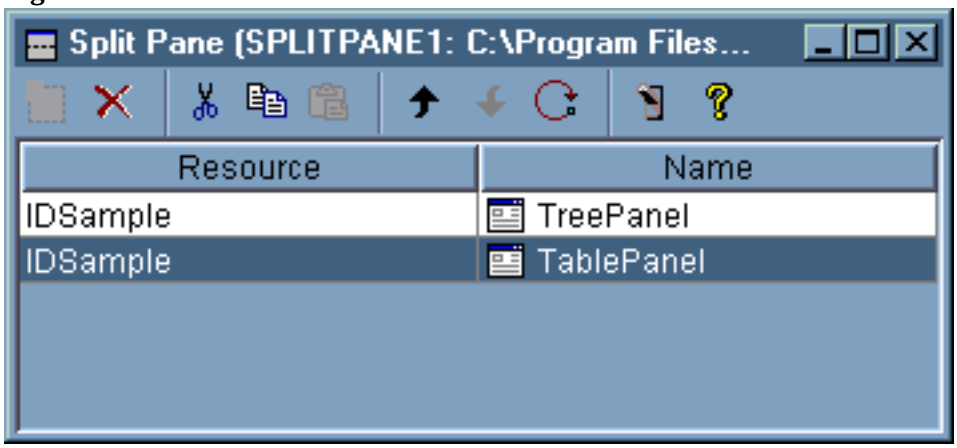


Crear una sección dividida con el Constructor de GUI

Con el Constructor de GUI resulta muy simple crear una sección dividida. En la barra de menús de la ventana principal del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

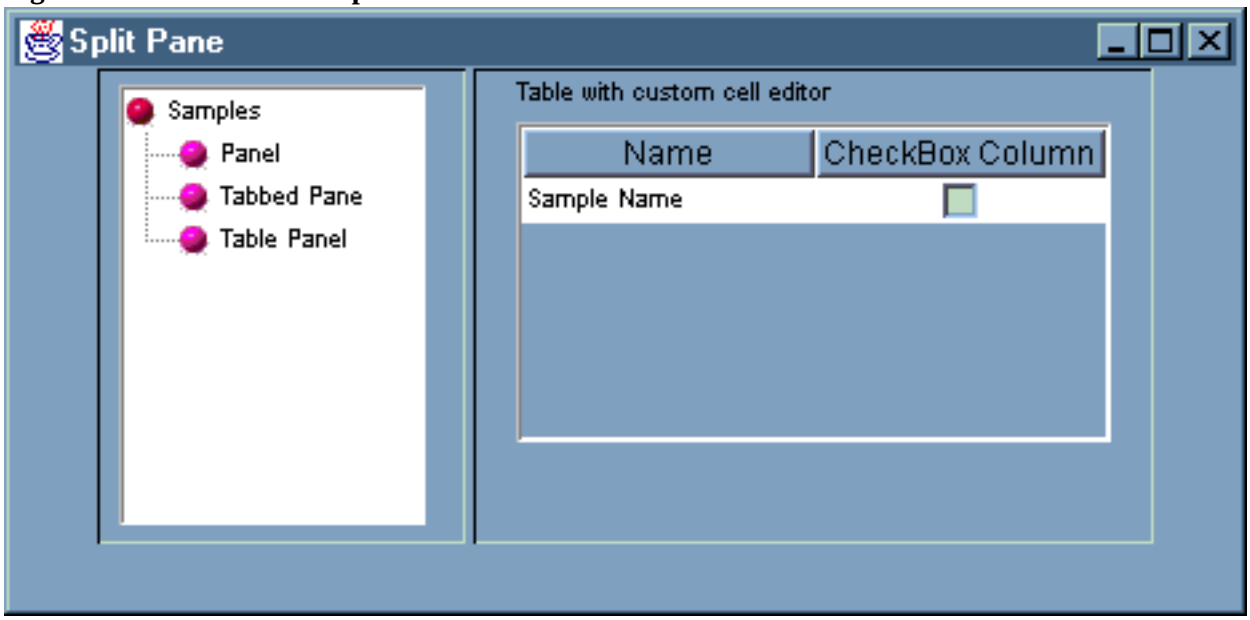
En la barra de menús de la ventana **Archivo** del Constructor de GUI, pulse el botón de la herramienta Insertar sección dividida  para visualizar un constructor de paneles donde puede insertar los componentes que desee para la sección dividida. En el ejemplo siguiente, se añaden dos componentes.

Figura 1: crear una sección dividida con el Constructor de GUI



Una vez creada la sección dividida, pulse el botón de la herramienta **Vista previa**  para obtener una vista previa de la misma, como se muestra en la figura 2.

Figura 2: obtener una vista previa de la sección dividida con el Constructor de GUI

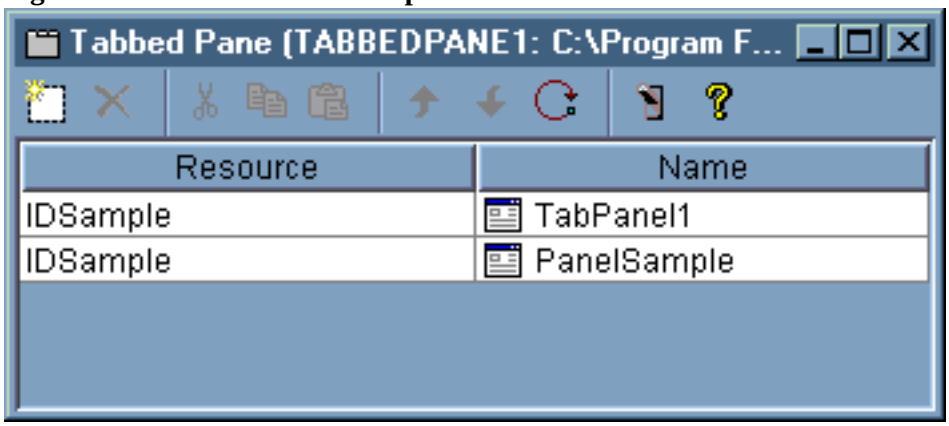


Crear una sección con pestañas con el Constructor de GUI

Con el Constructor de GUI es sencillo crear una sección con pestañas. En la barra de menús de la ventana principal del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

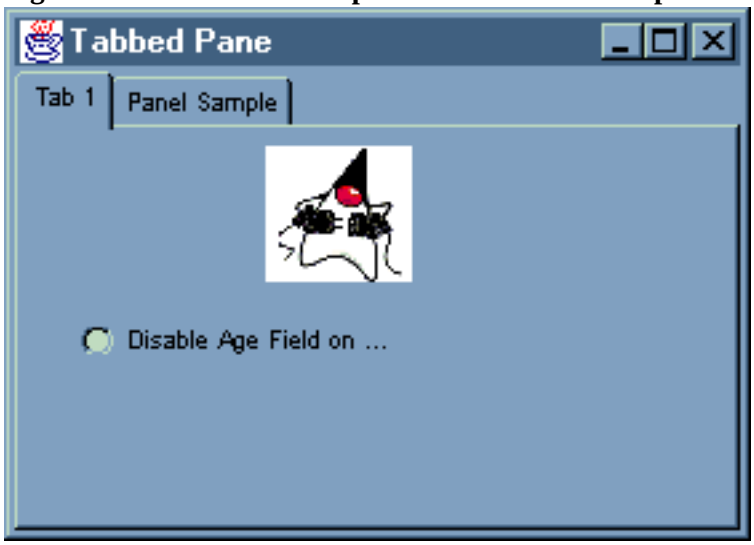
En la barra de menús de la ventana **Archivo** del Constructor de GUI, pulse el icono de Insertar sección con pestañas  para visualizar un constructor de paneles donde puede insertar los componentes que desee para la sección con pestañas. En el ejemplo siguiente, se añaden dos componentes.

Figura 1: crear una sección con pestañas en el Constructor de GUI



Una vez creada la sección con pestañas, pulse el botón de la herramienta **Vista previa**  para obtener una vista previa de la misma.

Figura 1: obtener una vista previa de la sección con pestañas con el Constructor de GUI

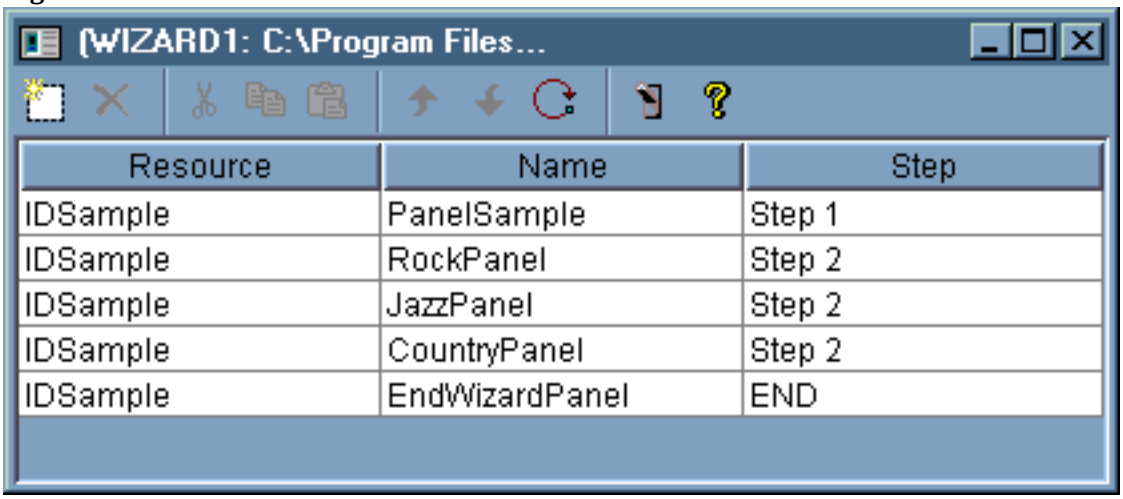


Crear un asistente con el Constructor de GUI

Con el Constructor de GUI es muy simple crear una interfaz de asistente. En la barra de menús de la ventana del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

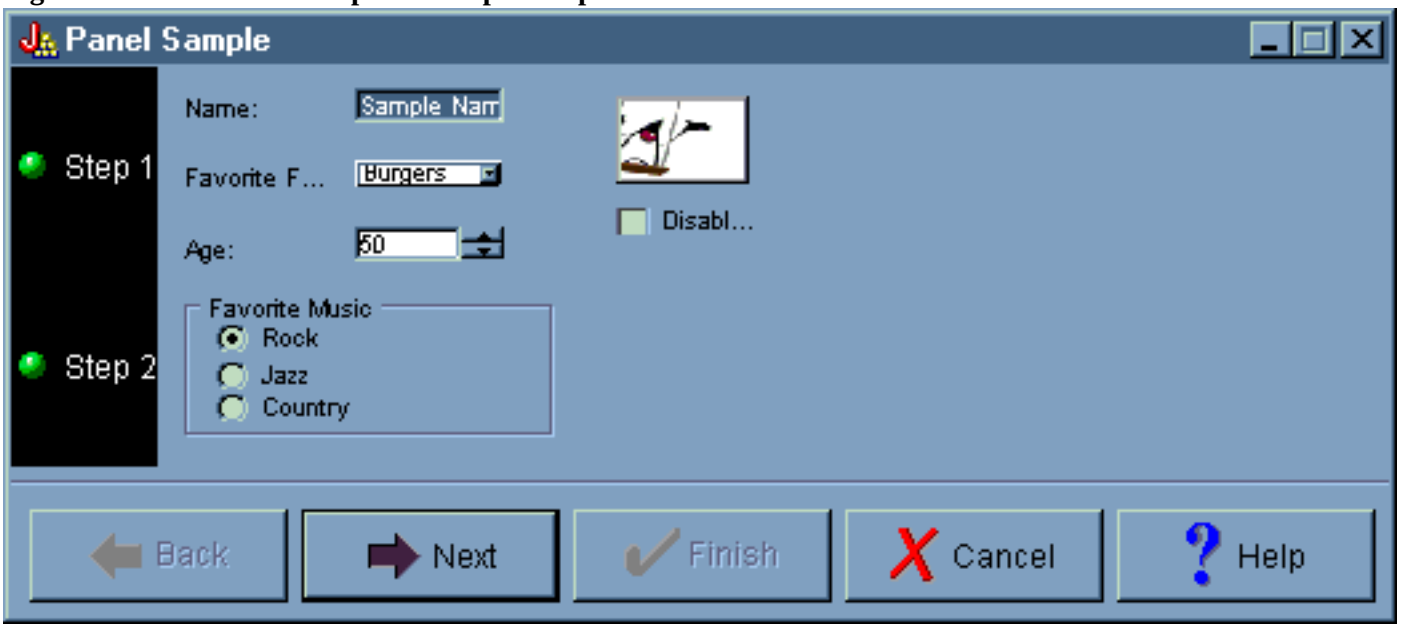
En la barra de menús de la ventana **Archivo** del Constructor de GUI, pulse el botón de la barra de herramientas de Insertar asistente  para visualizar un constructor de paneles donde puede añadir paneles al asistente.

Figura 1: crear un asistente con el Constructor de GUI



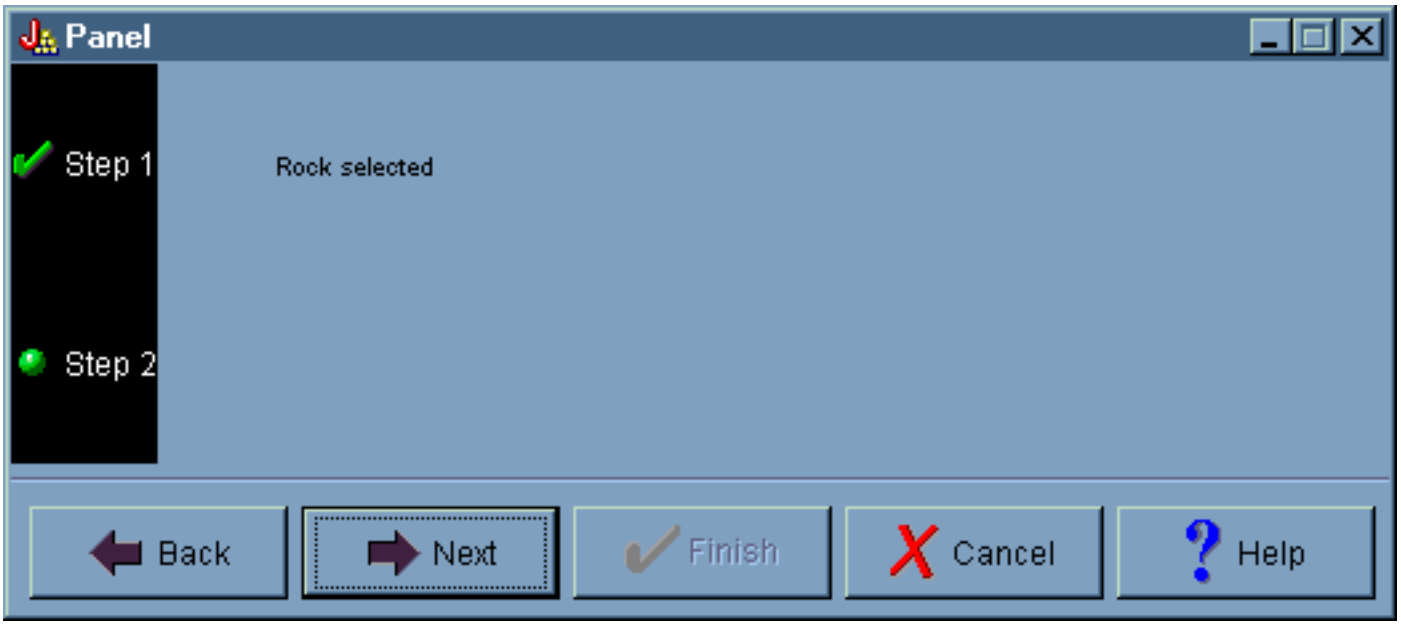
Una vez creado el asistente, utilice el botón de la herramienta **Vista previa**  para obtener una vista previa del mismo. La figura 2 muestra el panel que se visualiza en primer lugar para este ejemplo.

Figura 2: obtener una vista previa del primer panel del asistente con el Constructor de GUI



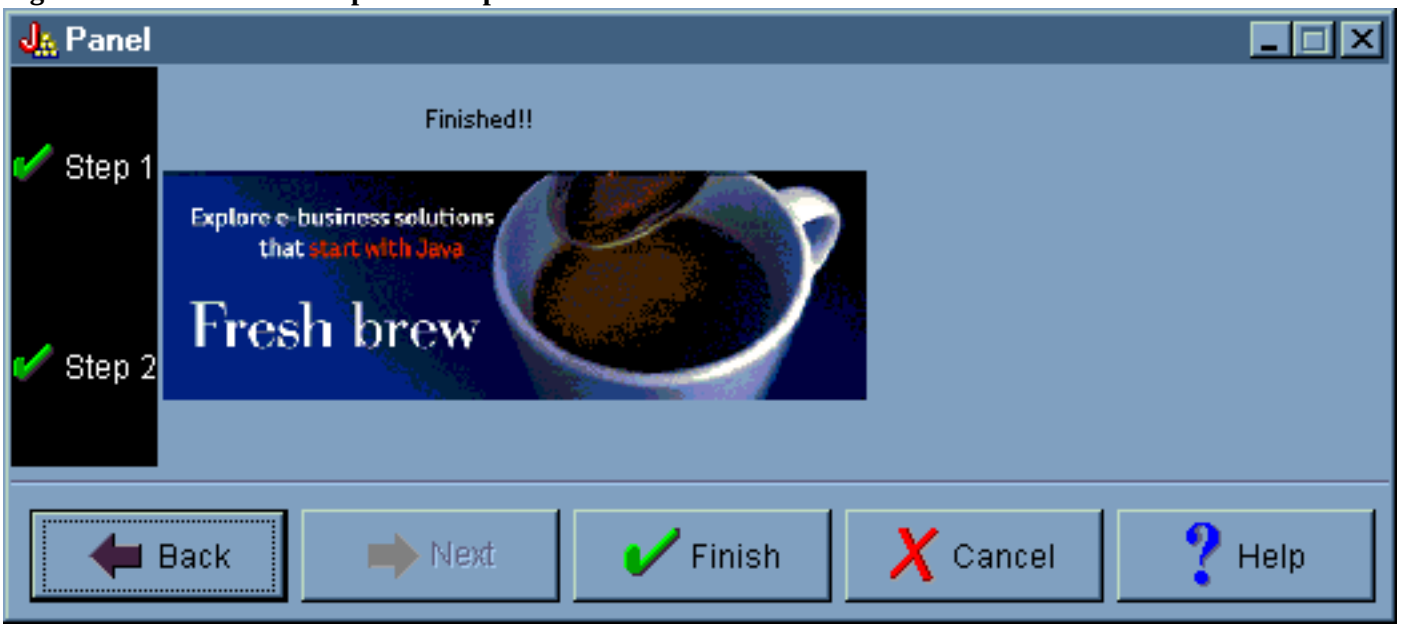
La figura 2 muestra el segundo panel que se visualiza cuando el usuario selecciona **Rock** y pulsa **Siguiente**.

Figura 2: obtener una vista previa del segundo panel del asistente con el Constructor de GUI



Al pulsar **Siguiente** en el segundo panel del asistente se visualiza el panel final del asistente, como se muestra en la figura 3.

Figura 3: obtener una vista previa del panel final del asistente con el Constructor de GUI

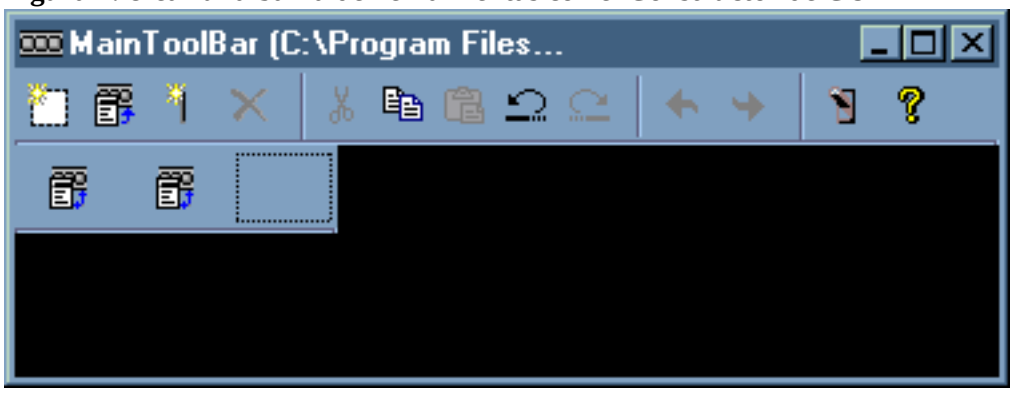


Crear una barra de herramientas con el Constructor de GUI

Con el Constructor de GUI es muy simple crear una barra de herramientas. En la barra de menús de la ventana del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

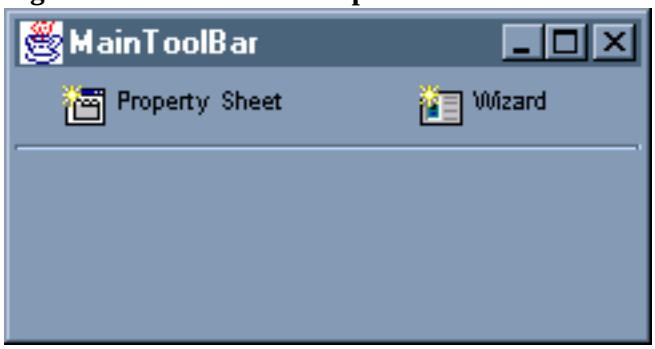
En la barra de menús de la ventana **Archivo** del Constructor de GUI, pulse el botón de la herramienta **Insertar barra de herramientas** para visualizar un constructor de paneles donde puede insertar los componentes que desee para la barra de herramientas.

Figura 1: crear una barra de herramientas con el Constructor de GUI



Una vez creada la barra de herramientas, pulse el botón de la herramienta **Vista previa**  para obtener una vista previa de la misma. Para este ejemplo, puede utilizar la barra de herramientas para visualizar una hoja de propiedades o un asistente.

Figura 2: obtener una vista previa de la barra de herramientas con el Constructor de GUI

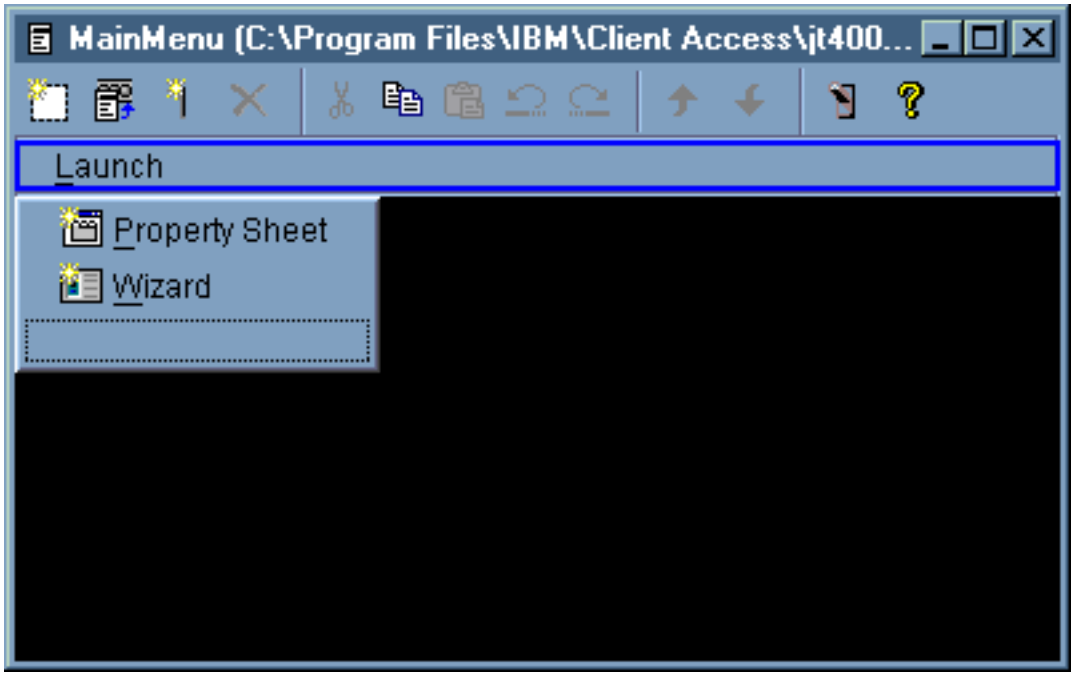


Crear una barra de menús con el Constructor de GUI

Con el Constructor de GUI es muy simple crear una barra de menús. En la barra de menús de la ventana del Constructor de GUI, seleccione **Archivo** --> **Archivo nuevo**.

En la barra de herramientas de la ventana **Archivo** del Constructor de GUI, pulse el botón de la herramienta **Insertar menú** para crear un constructor de paneles en el que puede insertar los componentes de que va a constar el menú.

Figura 1: crear un menú con el Constructor de GUI



Una vez creado el menú, utilice el botón de la herramienta **Vista previa**  para obtener una vista previa del mismo. Para este ejemplo, en el menú **Lanzar** que acaba de crear puede seleccionar **Hoja de propiedades** o **Asistente**. Las figuras siguientes ilustran lo que se muestra al seleccionar estos elementos de menú.

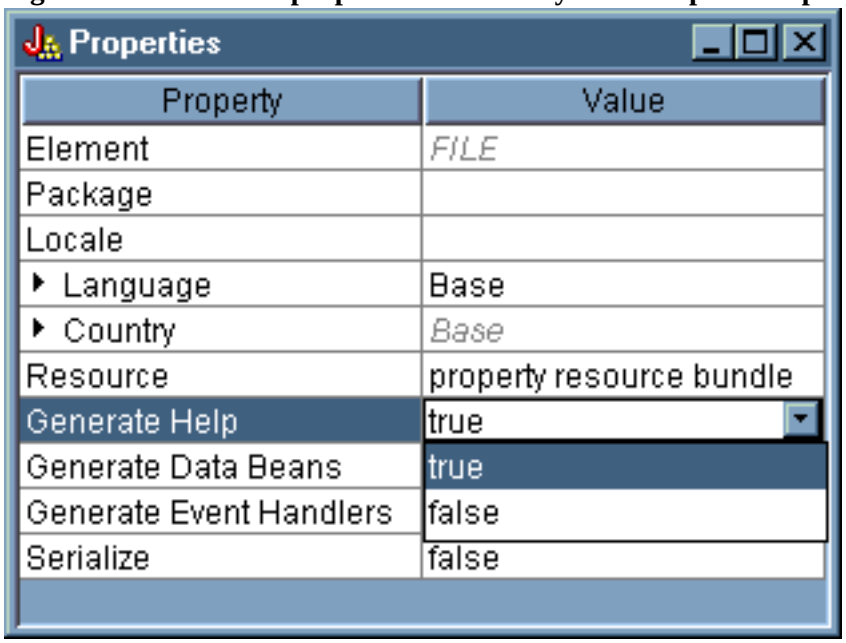
Figura 2: ver Hoja de propiedades en el menú Lanzar con el Constructor de GUI

Figura 3: ver Asistente en el menú Lanzar con el Constructor de GUI

Ejemplo: crear el documento de ayuda

Crear archivos de ayuda con el Constructor de GUI es una tarea simple. En el panel de propiedades del archivo con el que está trabajando, establezca "Generar ayuda" en verdadero (true).

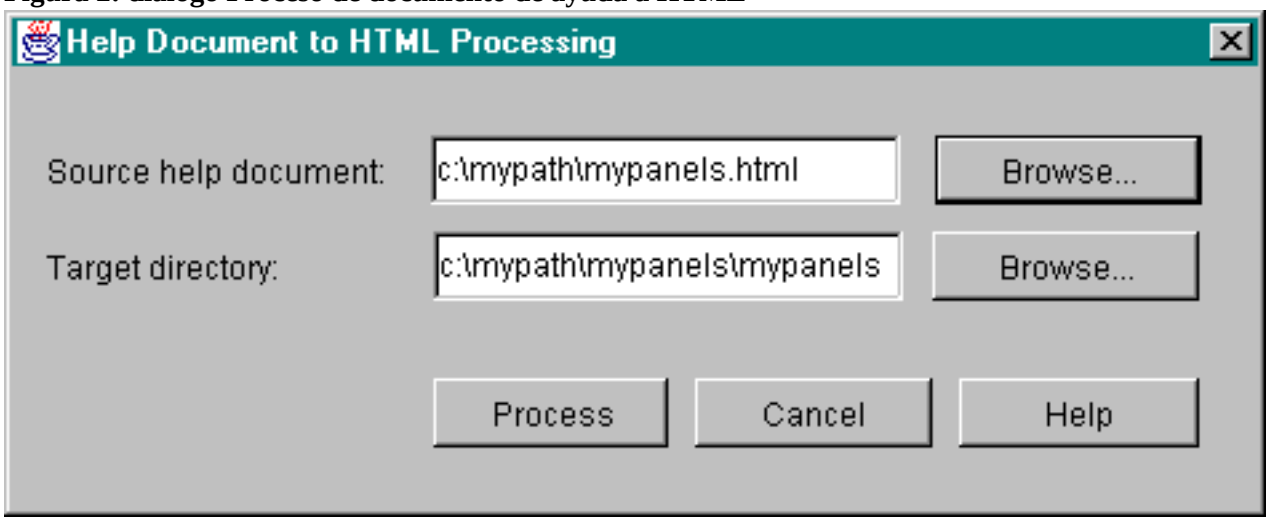
Figura 1: establecer la propiedad Generar ayuda en el panel de propiedades del Constructor de GUI



El Constructor de GUI crea una infraestructura HTML llamada documento de ayuda que se puede editar.

Los temas que hay en el archivo PDML, para que se puedan utilizar en tiempo de ejecución, se deben separar en archivos HTML individuales. Cuando se ejecuta el **Proceso de documento de ayuda a HTML**, los temas se desglosan en archivos individuales y se ponen en un subdirectorio que recibe el nombre del documento de ayuda y del archivo PDML. El entorno de ejecución prevé que los archivos HTML individuales estén en un subdirectorio que tenga el mismo nombre que el documento de ayuda y el archivo PDML. El diálogo **Proceso de documento de ayuda a HTML** reúne la información necesaria y llama al programa HelpDocSplitter para hacer el proceso:

Figura 2: diálogo Proceso de documento de ayuda a HTML



El proceso de documento de ayuda a HTML se inicia escribiendo lo siguiente en una solicitud de mandatos:

```
jre com.ibm.as400.ui.tools.hdoc2htmViewer
```

Para ejecutar este mandato es preciso que la vía de acceso de clases esté debidamente configurada.

Para utilizar el proceso de documento de ayuda a HTML, primero se selecciona el documento de ayuda que tiene el mismo nombre que el archivo PDML. Después se especifica para la salida un subdirectorio que tenga el mismo nombre que el documento de ayuda y que el archivo PDML. Por último, se selecciona "Procesar" para completar el proceso.

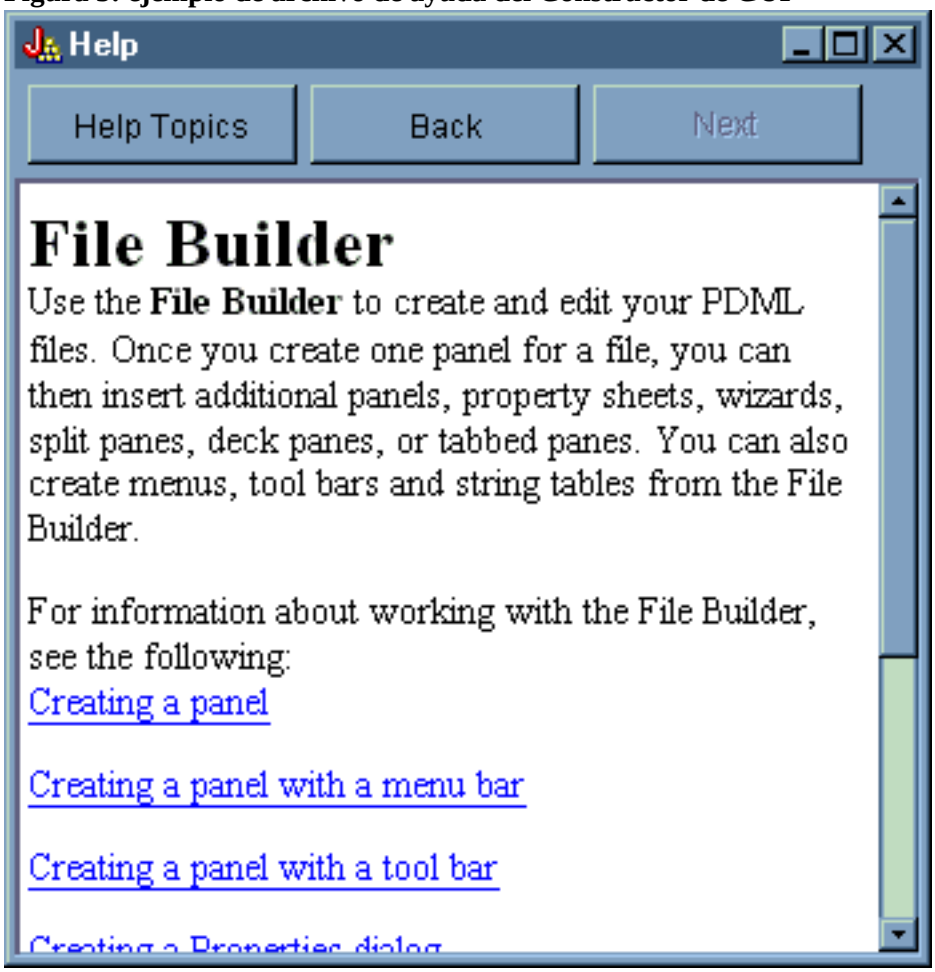
Puede subdividir el documento de ayuda desde la línea de mandatos con este mandato:

```
jre com.ibm.as400.ui.tools.HelpDocSplitter "helpdocument.htm" [directorio salida]
```

Este mandato ejecuta el proceso que desglosa el archivo. Como entrada se proporciona el nombre del documento de ayuda junto con un directorio de salida opcional. Por omisión, se crea un subdirectorio que tiene el mismo nombre que el documento de ayuda y en él se ponen los archivos resultantes.

Este es un ejemplo del aspecto que puede tener un archivo de ayuda:

Figura 3: ejemplo de archivo de ayuda del Constructor de GUI

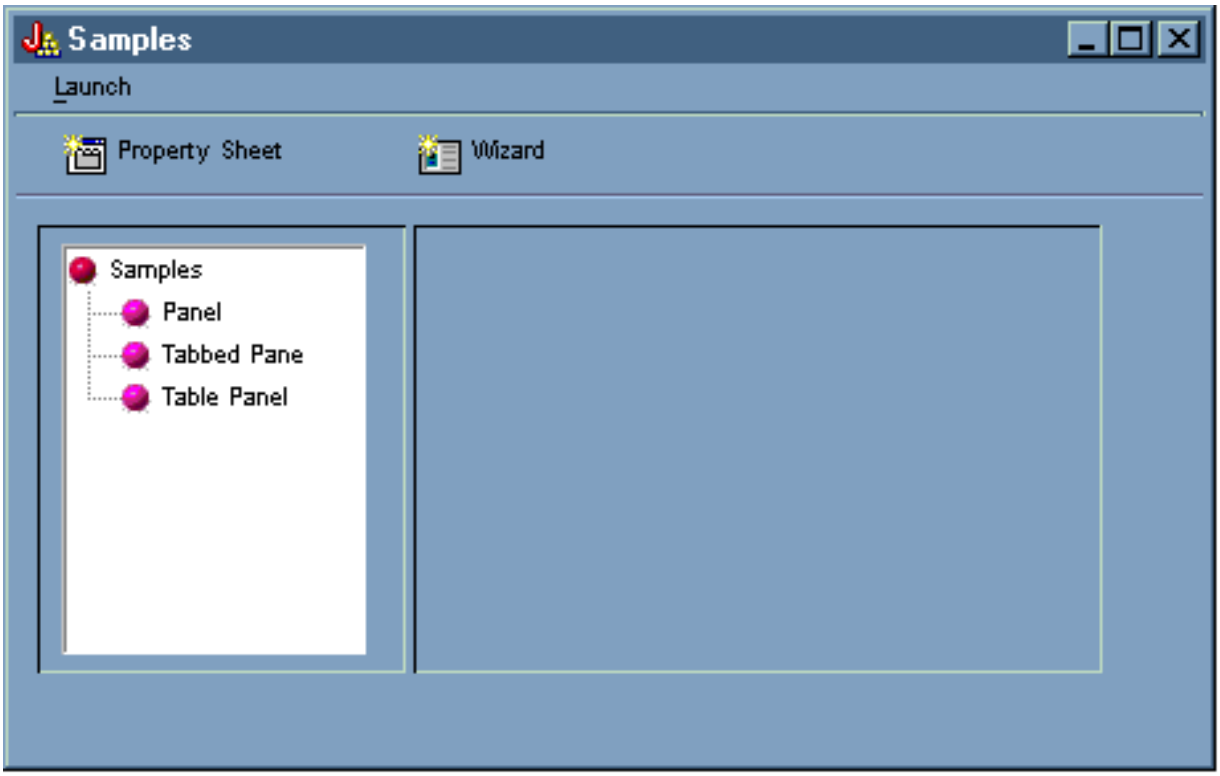


Ejemplo: cómo se utiliza el Constructor de GUI

Cuando se reúnen los ejemplos contenidos en esta sección mientras funcionan internamente los beans de datos correctos, se obtiene una aplicación GUI total.

La figura 1 muestra el primer panel que aparece al ejecutar este ejemplo.

Figura 1: ventana principal del ejemplo de Constructor de GUI



Observe que esta pantalla le permite utilizar el gestor de paneles dinámico. Las figuras 2 y 3 muestran cómo puede ajustar el tamaño de la ventana para hacerla más grande o más pequeña:

Figura 2: cómo se ajusta el tamaño de la ventana principal del ejemplo de Constructor de GUI (ampliación)

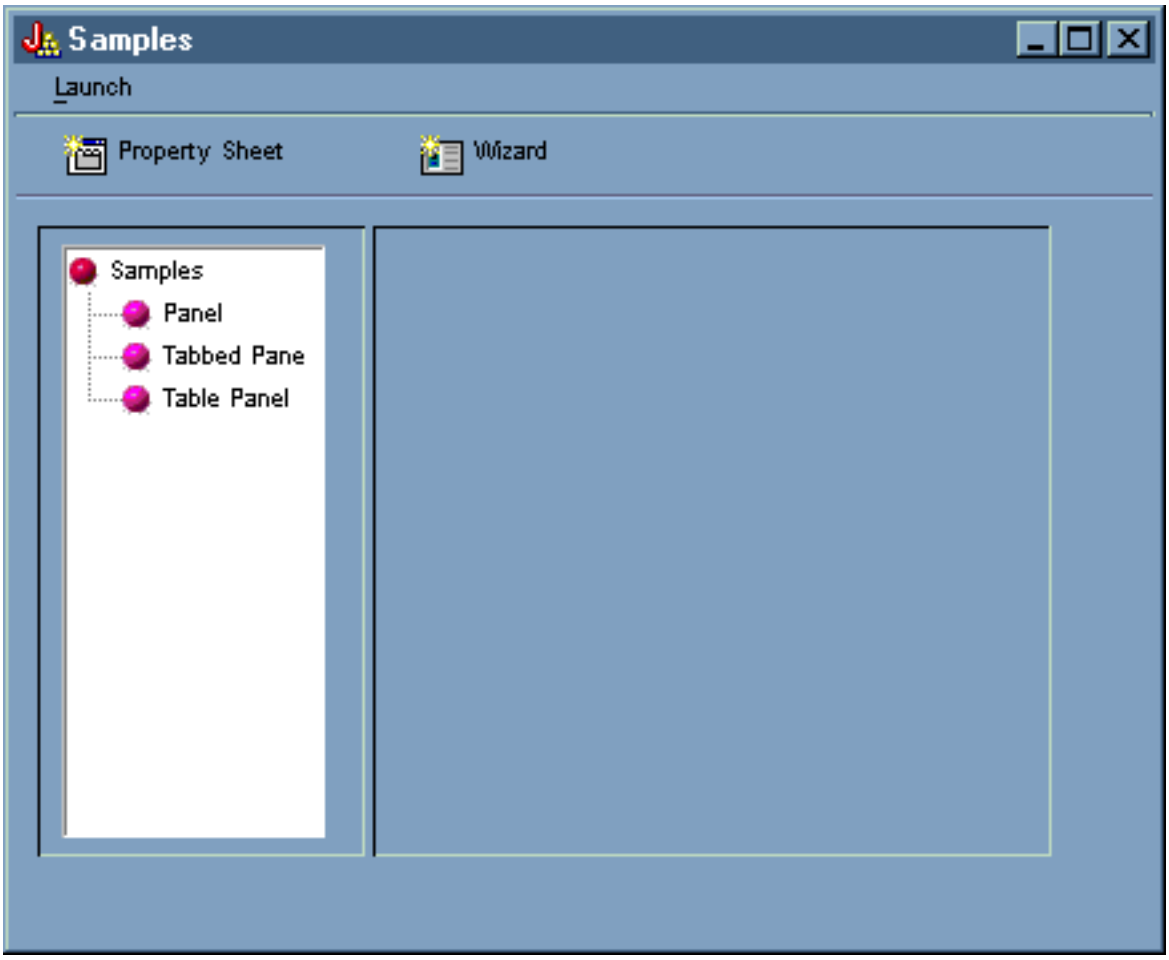
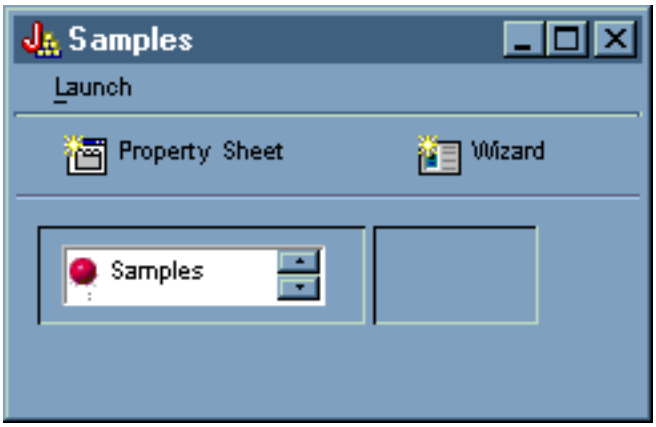


Figura 3: cómo se ajusta el tamaño de la ventana principal del ejemplo de Constructor de GUI (reducción)



Al utilizar el gestor de paneles dinámico, aunque cambie el tamaño del panel y de los controles del panel, el tamaño del texto no varía.

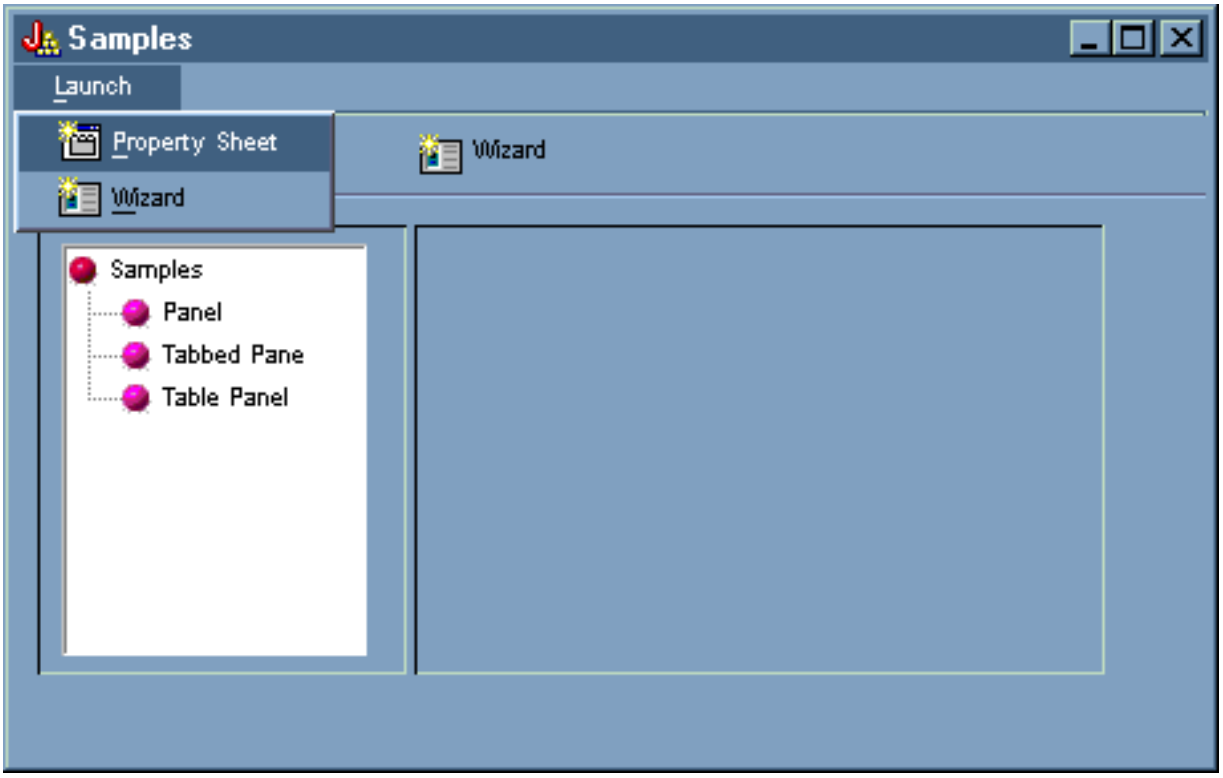
El panel permite llevar a cabo las acciones siguientes:

- Iniciar una hoja de propiedades
- Iniciar un asistente
- Mostrar los ejemplos listados en la sección izquierda

Iniciar la hoja de propiedades

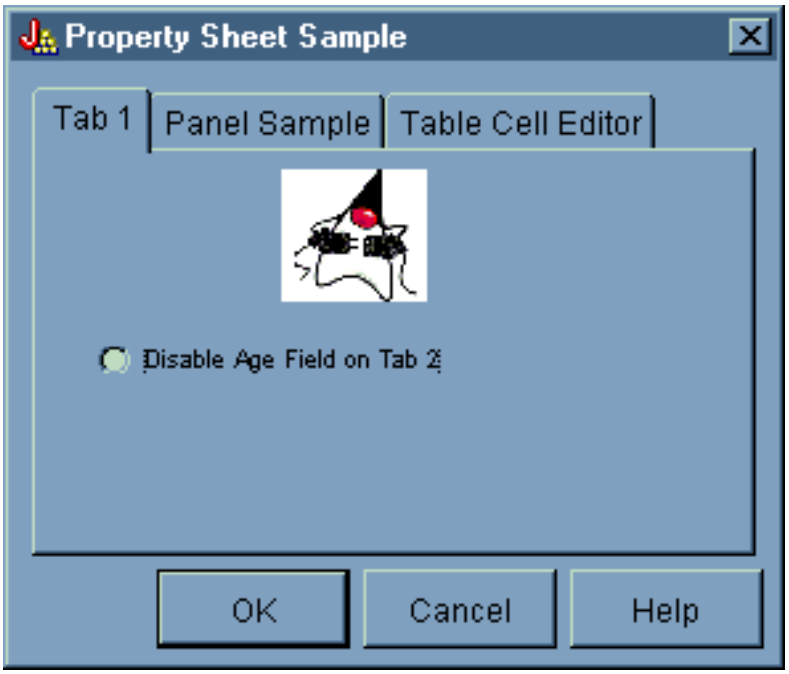
Puede iniciar la hoja de propiedades pulsando el botón de la barra de herramientas Hoja de propiedades o mediante el menú **Lanzar**. El hecho de poder elegir entre la barra de herramientas y el menú ilustra el enlace de los elementos de menú. La figura 4 muestra la selección de **Hoja de propiedades** en el menú **Lanzar** en la ventana principal del ejemplo de Constructor de GUI.

Figura 4: seleccionar Hoja de propiedades desde el menú Lanzar



Al seleccionar **Hoja de propiedades** se visualiza el panel de la figura 5.

Figura 5: diálogo de ejemplo de hoja de propiedades



Cuando aparece por primera vez el ejemplo de hoja de propiedades, se visualiza la pestaña 1 por omisión. Las figuras 6 y 7 muestran cómo cambia la visualización del panel cuando se seleccionan las demás pestañas.

Figura 6: seleccionar la pestaña Ejemplo de panel

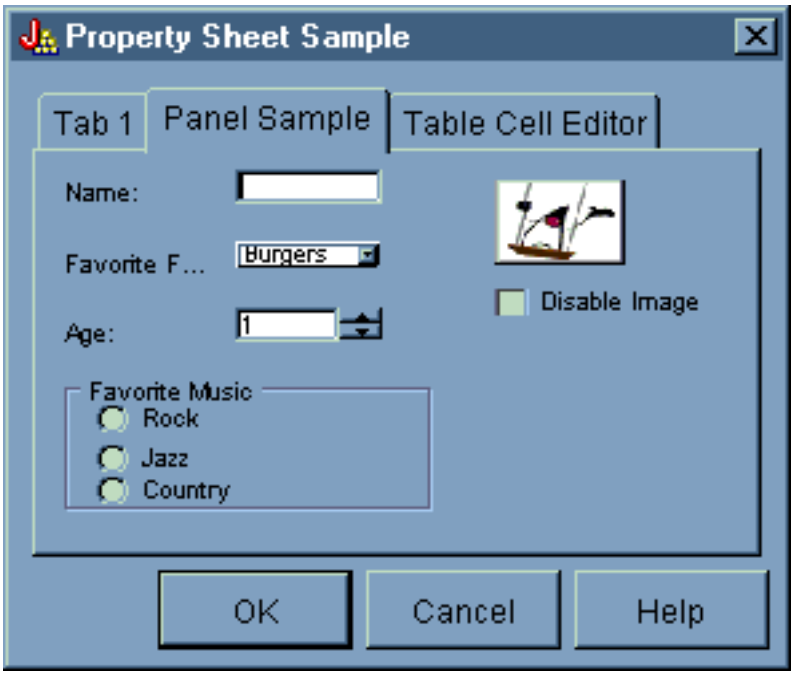
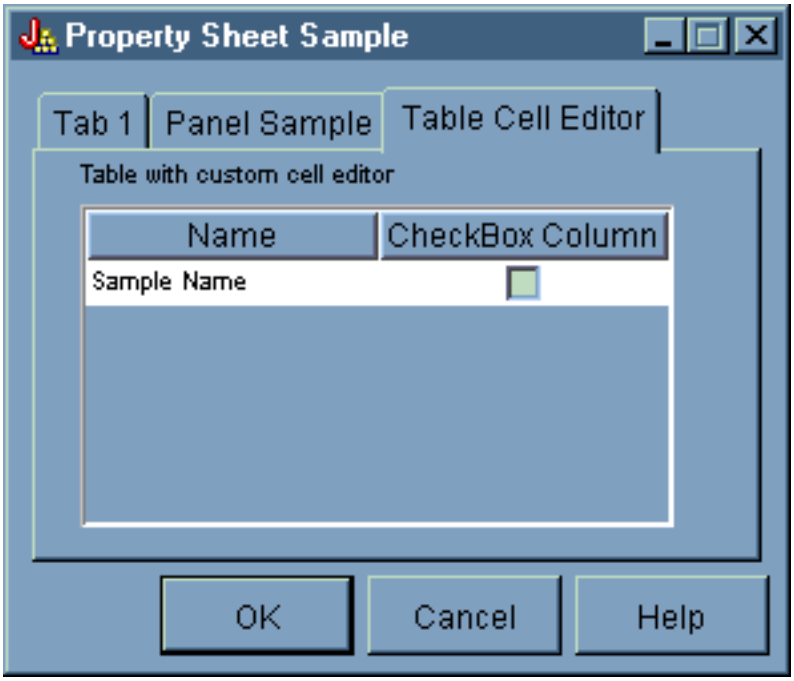


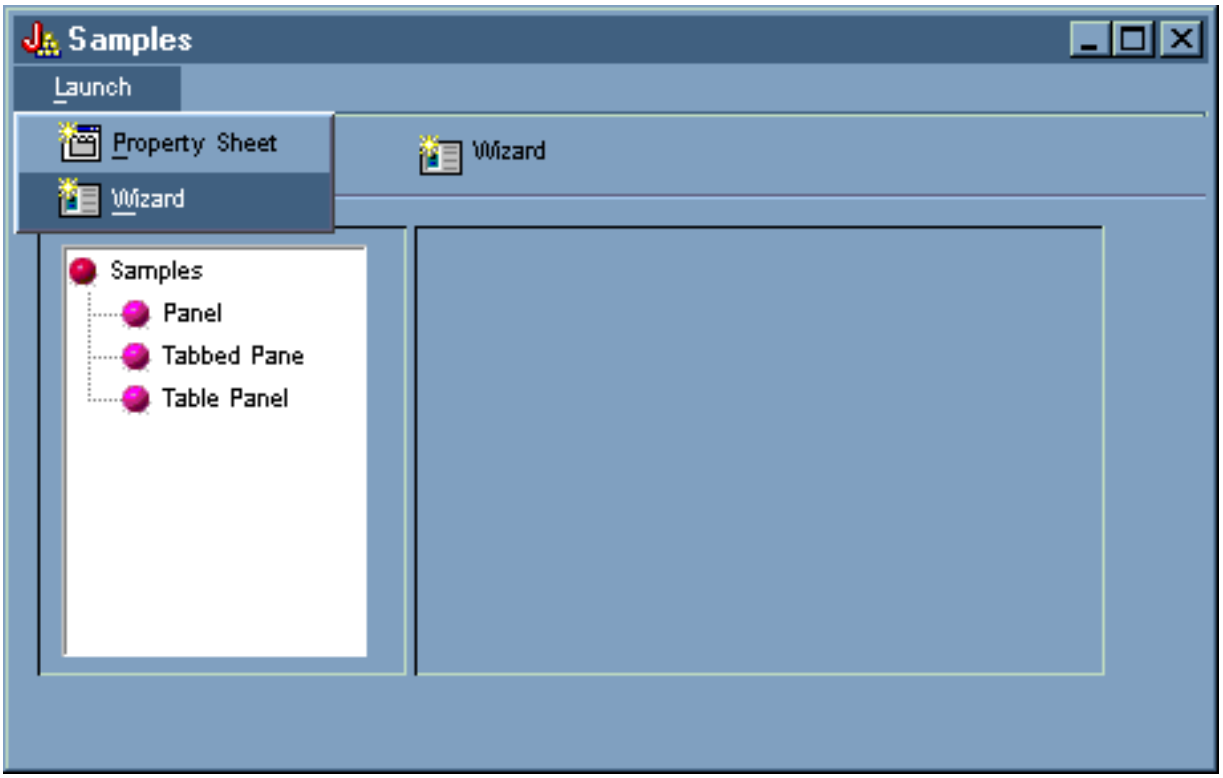
Figura 7: seleccionar la pestaña de editor de casillas de tabla



Iniciar el asistente

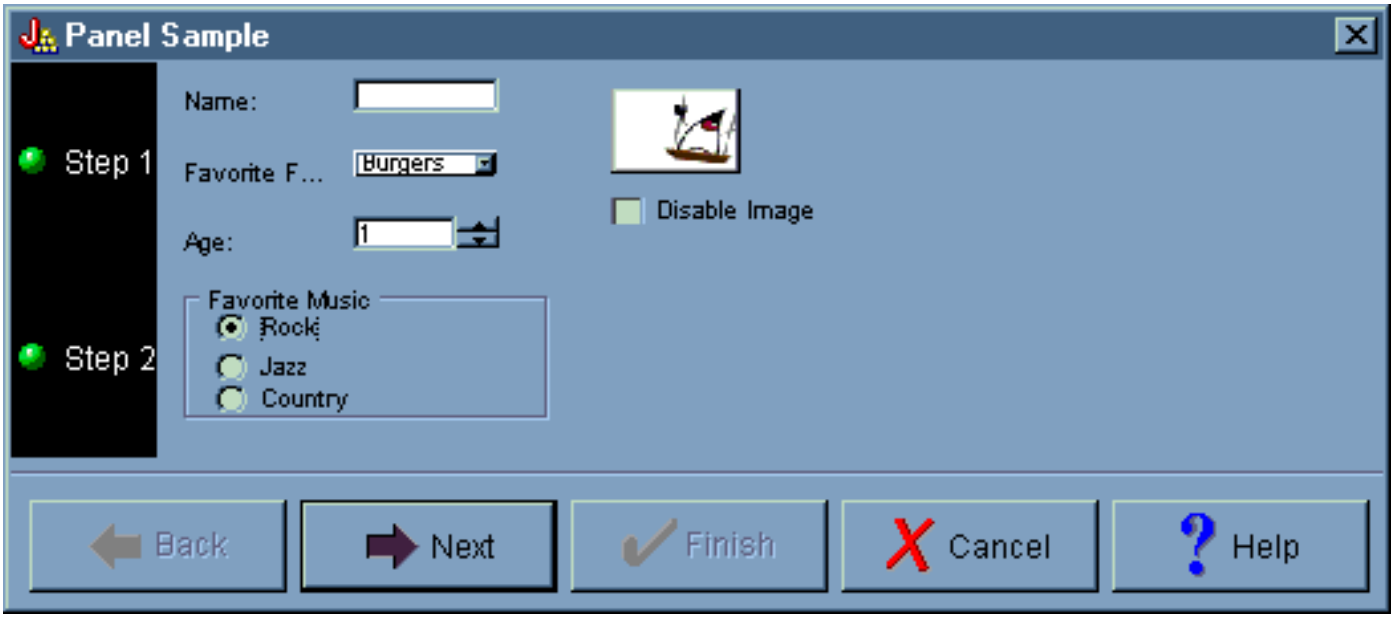
Puede iniciar el asistente pulsando el botón de la barra de herramientas Asistente o mediante el menú **Lanzar**. El hecho de poder elegir entre la barra de herramientas y el menú ilustra el enlace de los elementos de menú. La figura 8 muestra la selección de **Asistente** en el menú **Lanzar** en la ventana principal del ejemplo de Constructor de GUI.

Figura 8: seleccionar Asistente desde el menú Lanzar



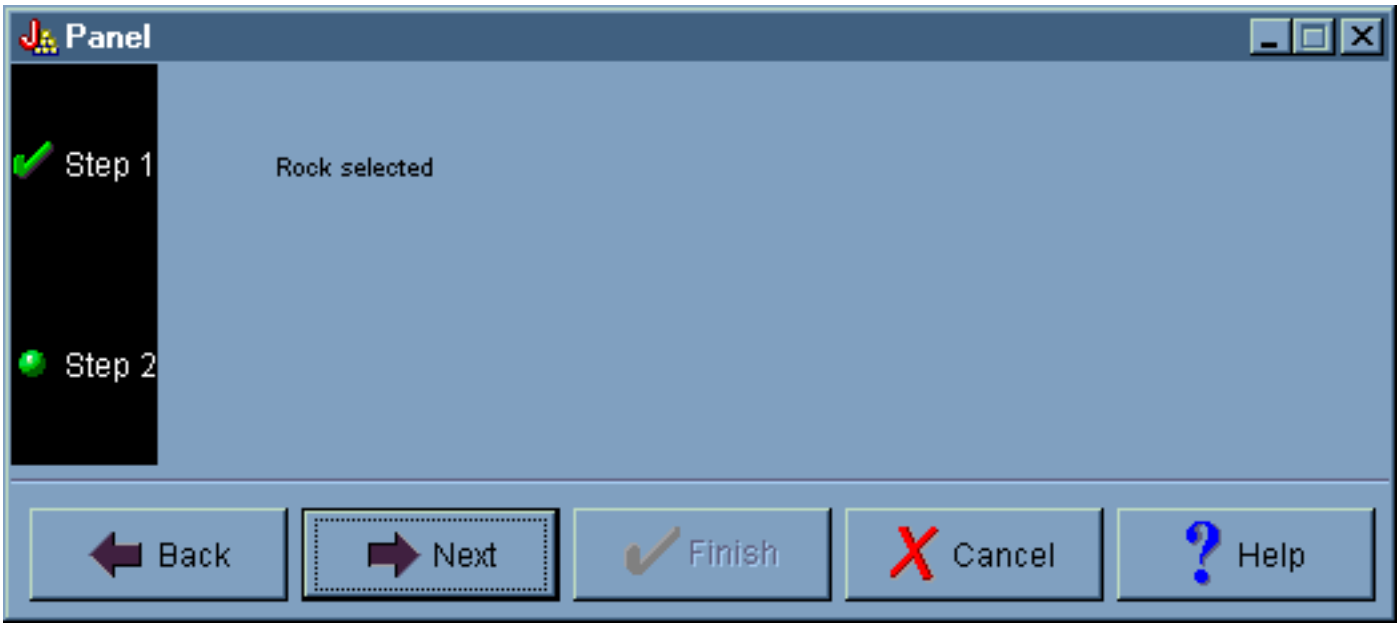
La figura 9 muestra las numerosas opciones que proporciona el primer diálogo del asistente.

Figura 9: seleccionar Rock en el primer diálogo del asistente



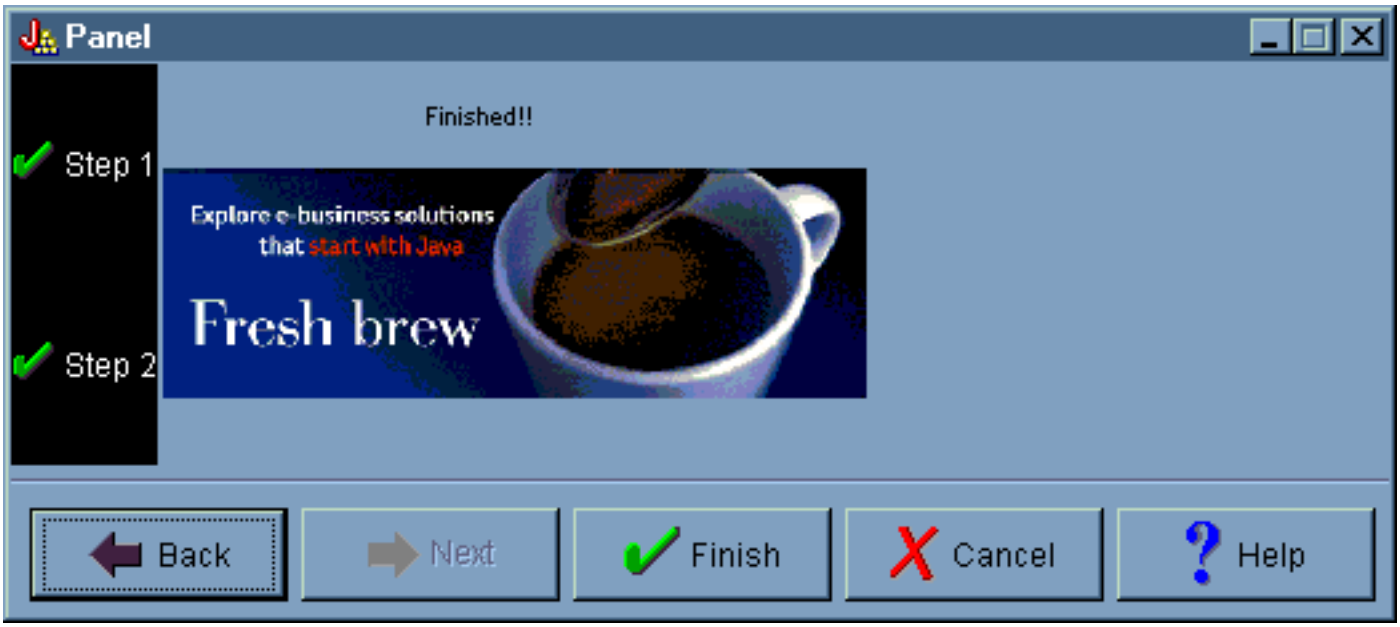
En el primer diálogo del asistente, seleccione **Rock** y pulse **Siguiente** para visualizar el segundo diálogo del asistente como se muestra en la figura 10.

Figura 10: segundo diálogo del asistente (tras seleccionar Rock)



En el segundo diálogo del asistente, pulse **Siguiente** para visualizar el diálogo final del asistente como se muestra en la figura 11.

Figura 11: diálogo final del asistente



Sin embargo, este ejemplo se ha programado para repetirse en bucle. Seleccione **Country** en el primer diálogo del asistente (figura 12) y, a continuación, pulse **Siguiente** para visualizar el segundo diálogo del asistente (figura 13). Al pulsar Siguiente en el segundo diálogo del asistente se visualiza de nuevo el primer diálogo (figura 14) en lugar del diálogo final del asistente.

Figura 12: seleccionar Country en el primer diálogo del asistente

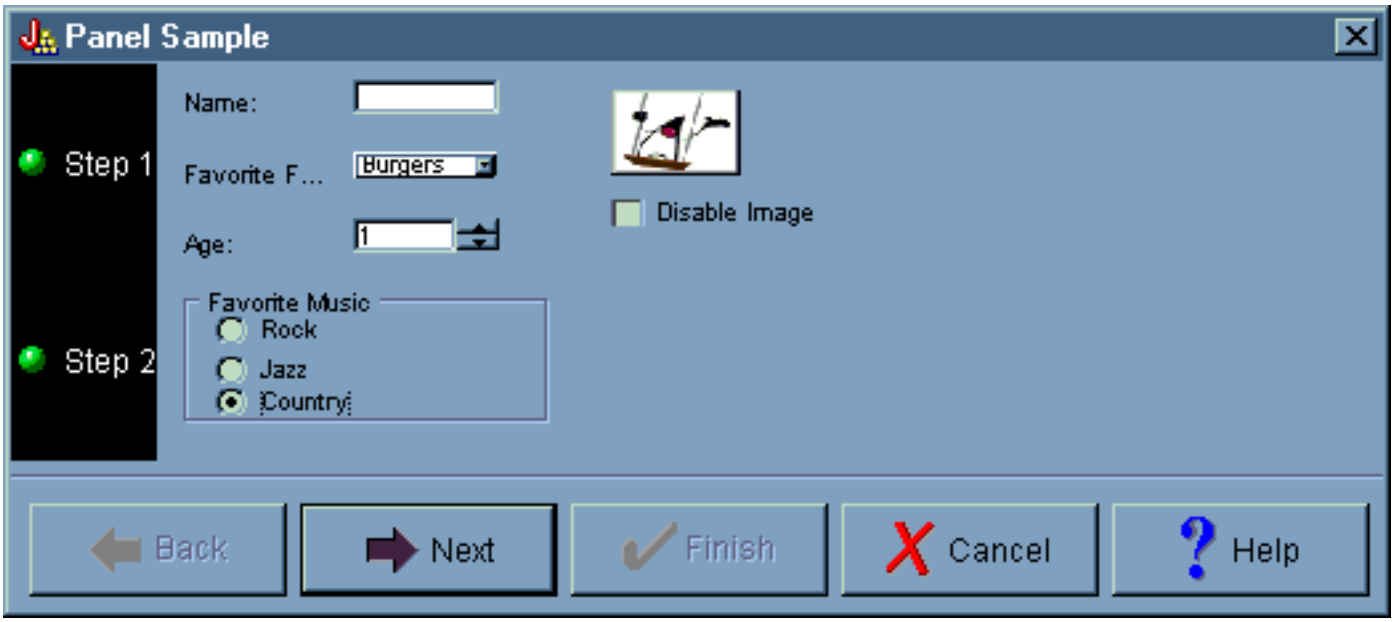


Figura 13: segundo diálogo del asistente (tras seleccionar Country)

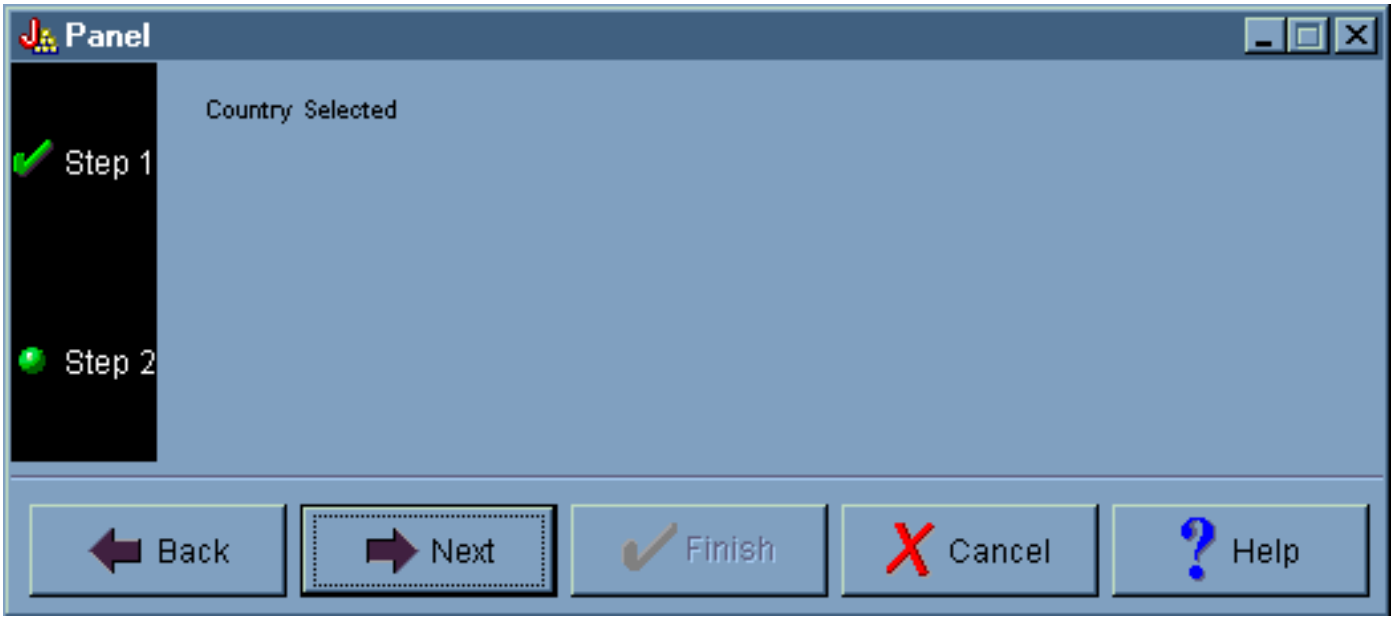
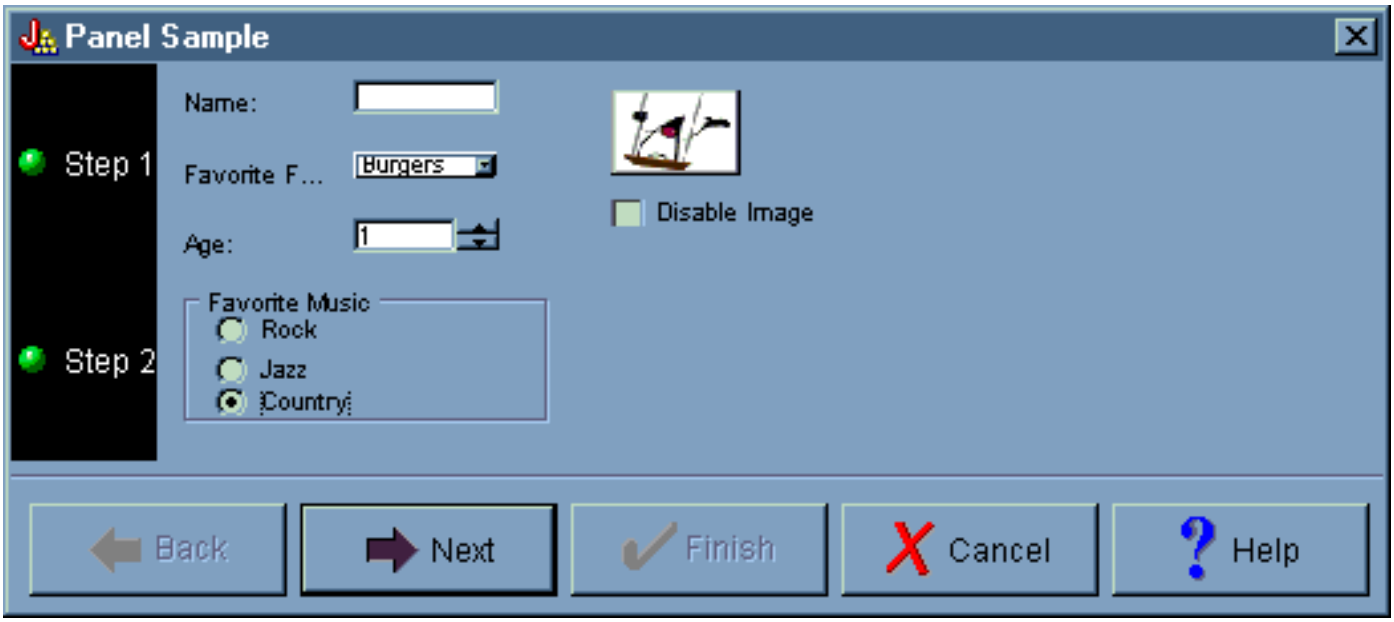


Figura 14: volver al primer diálogo del asistente

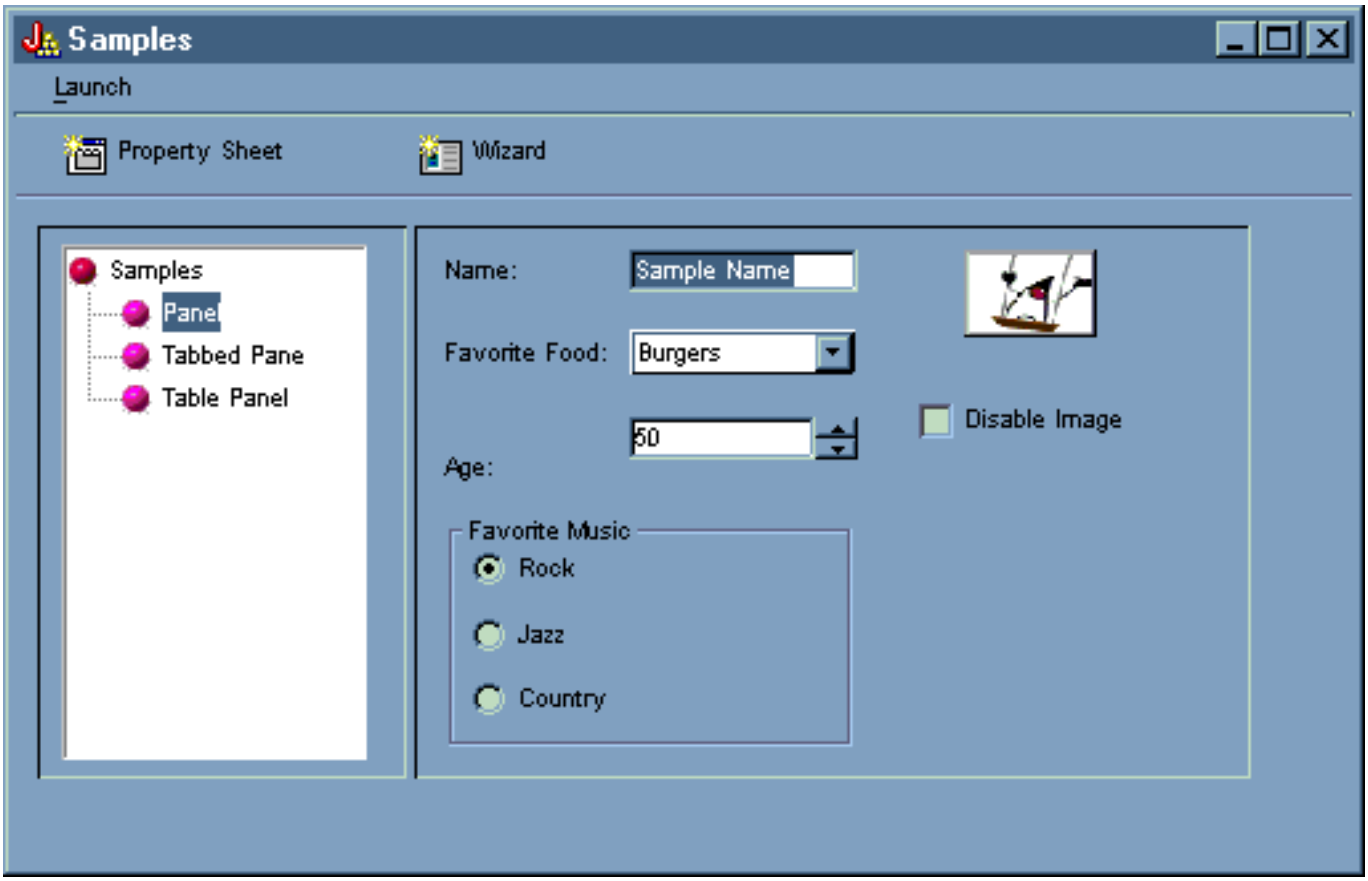


En otras palabras, el programador ha determinado que nadie debe seleccionar que su música favorita sea la música country.

Visualizar los ejemplos

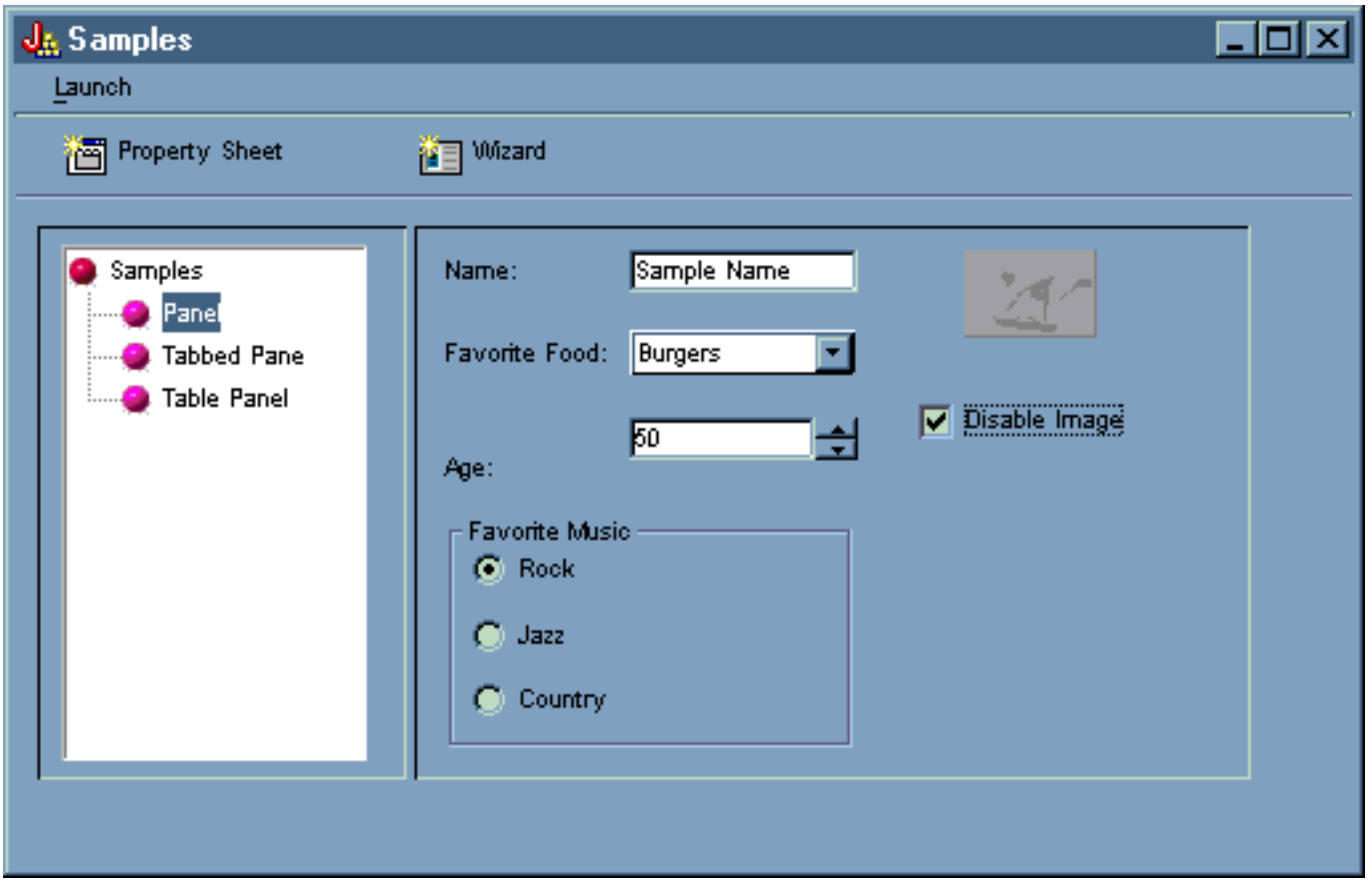
En la ventana principal del ejemplo de Constructor de GUI también puede seleccionar otras funciones en la sección izquierda situada debajo de la barra de herramientas. La figura 15 muestra cómo al seleccionar **Panel** en la sección izquierda se visualiza el ejemplo de panel en la sección derecha.

Figura 15: seleccionar Panel en la sección izquierda



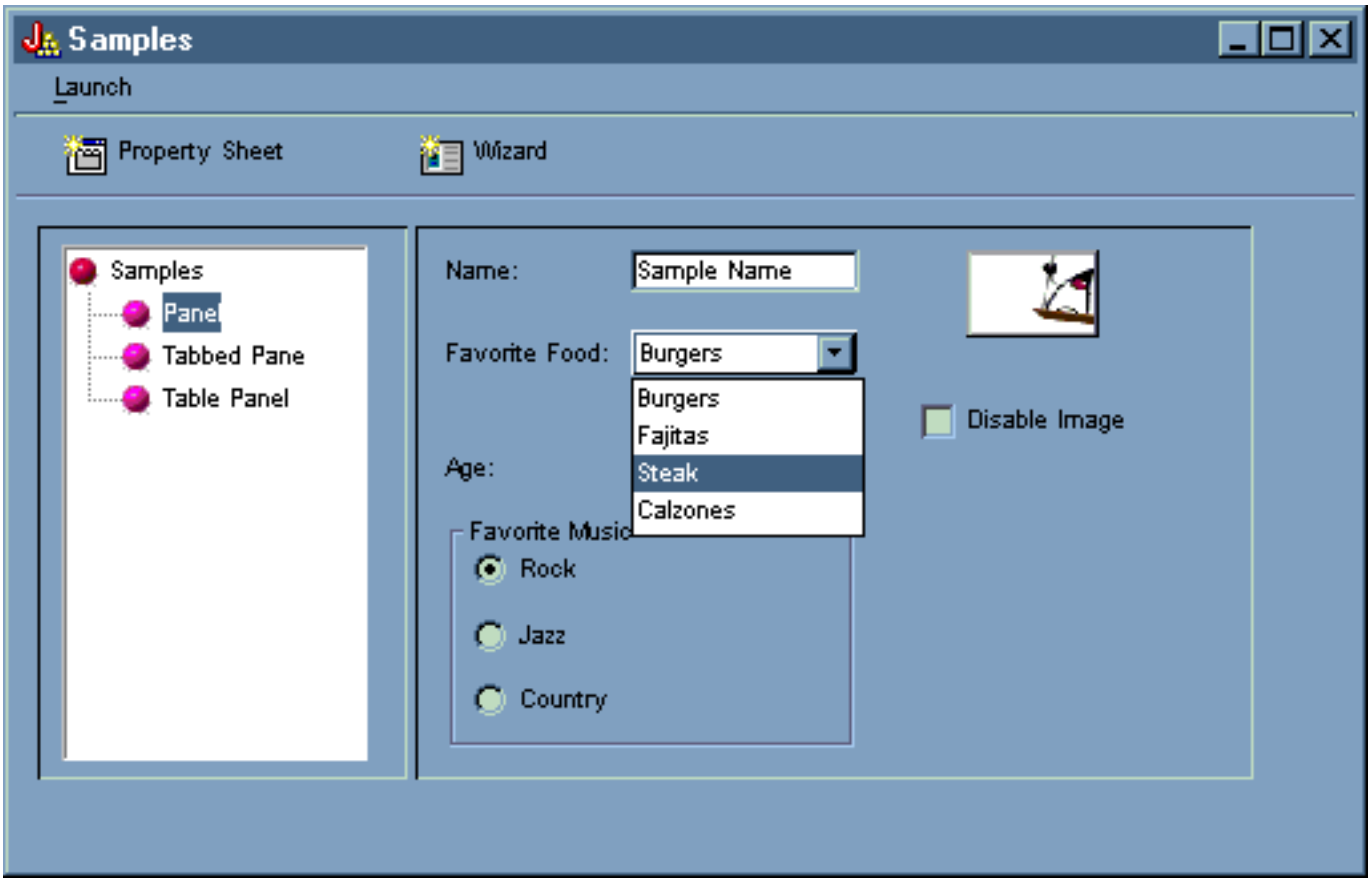
El ejemplo de panel se ha programado con una opción para inhabilitar la imagen. Seleccione **Inhabilitar imagen** para ver la misma pantalla pero con la imagen atenuada, como se muestra en la figura 16.

Figura 16: seleccionar Inhabilitar imagen en la sección derecha



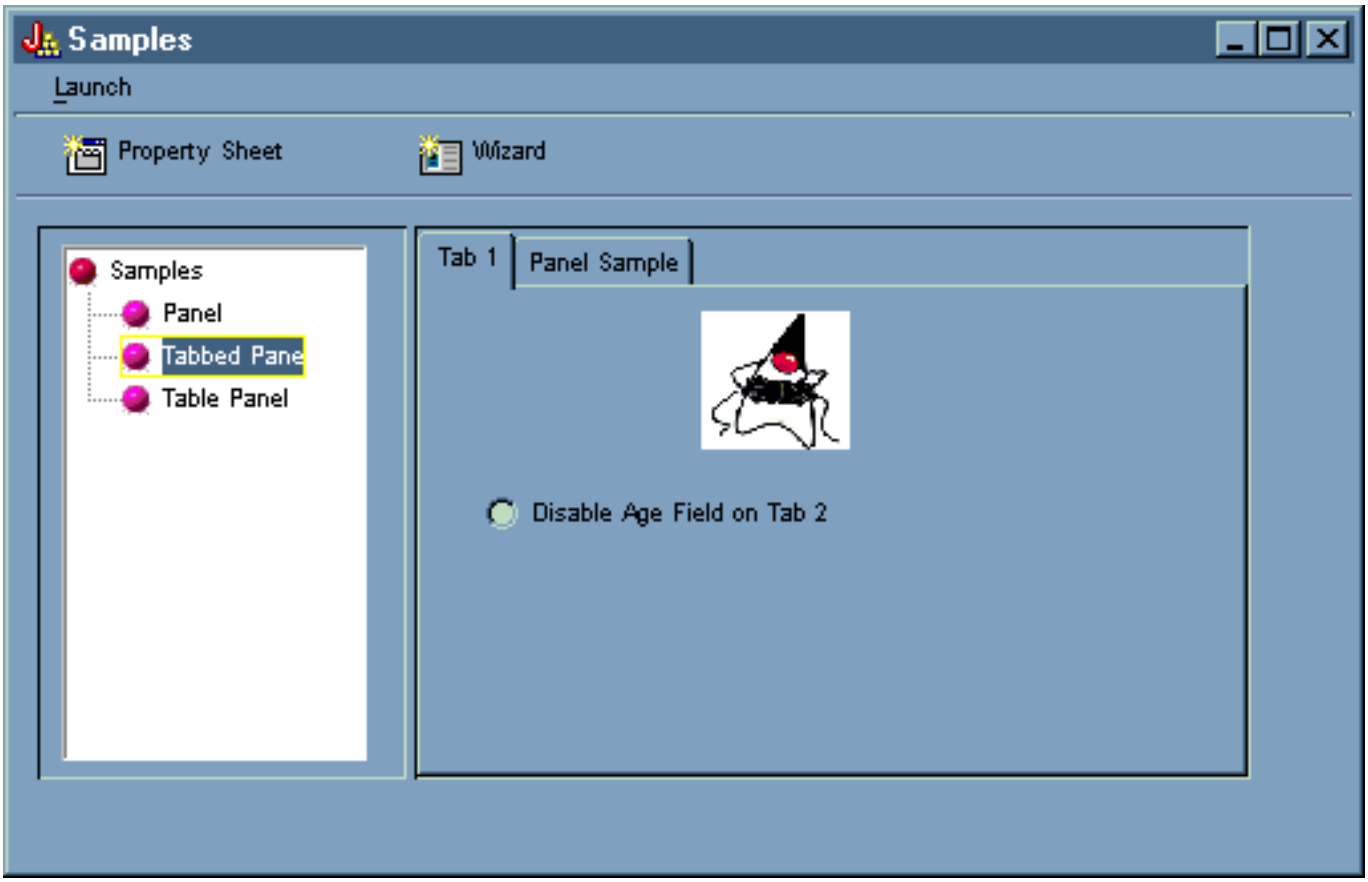
El ejemplo de panel también tiene la opción de recuadro de lista desplegable, como se muestra en la figura 17.

Figura 17: seleccionar un elemento de la lista Alimentos favoritos en la sección derecha



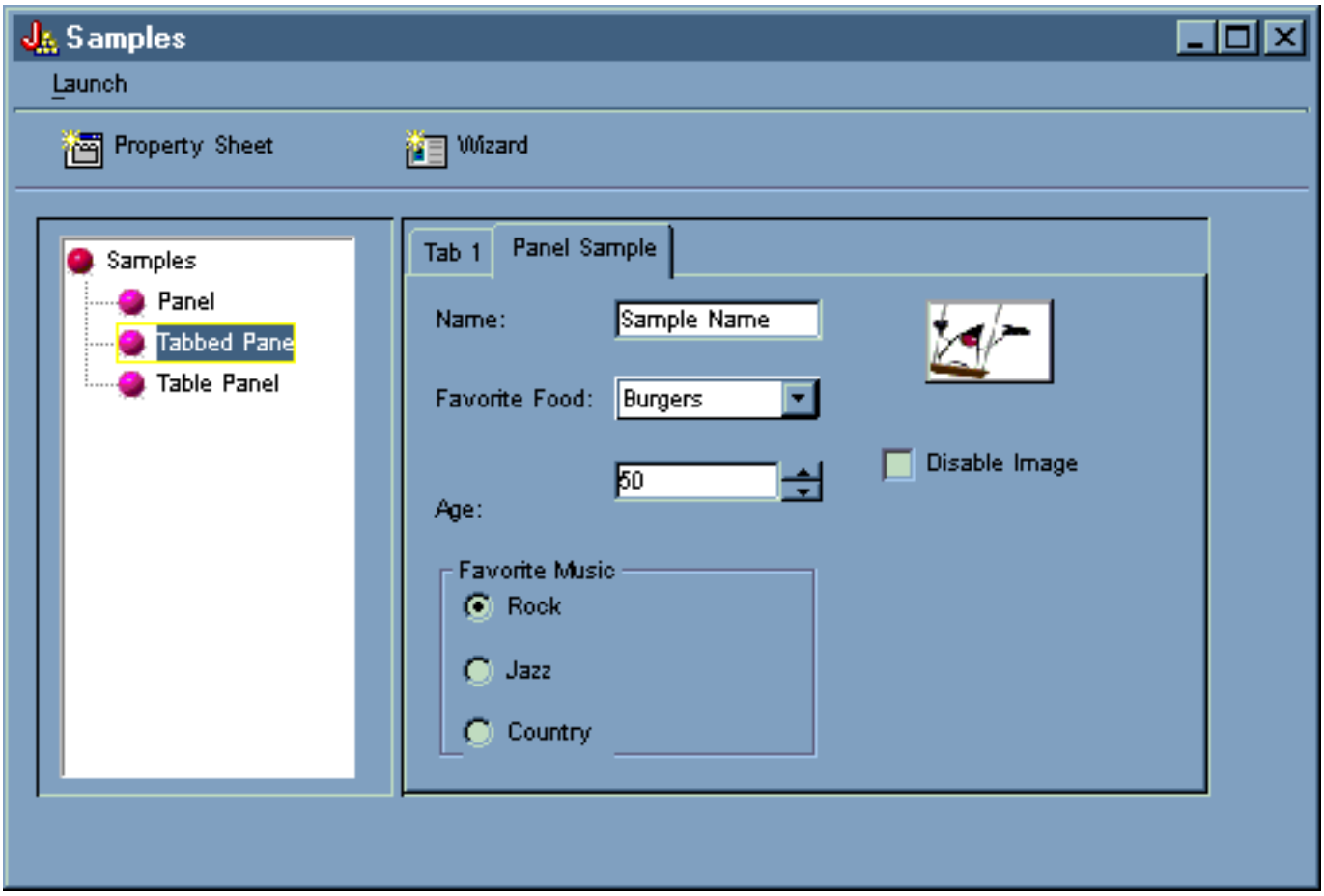
La figura 18 muestra cómo al seleccionar **Sección con pestañas** en la sección izquierda de la ventana principal del ejemplo de Constructor de GUI se visualiza el ejemplo de sección con pestañas en la sección derecha.

Figura 18: seleccionar Sección con pestañas en la sección izquierda



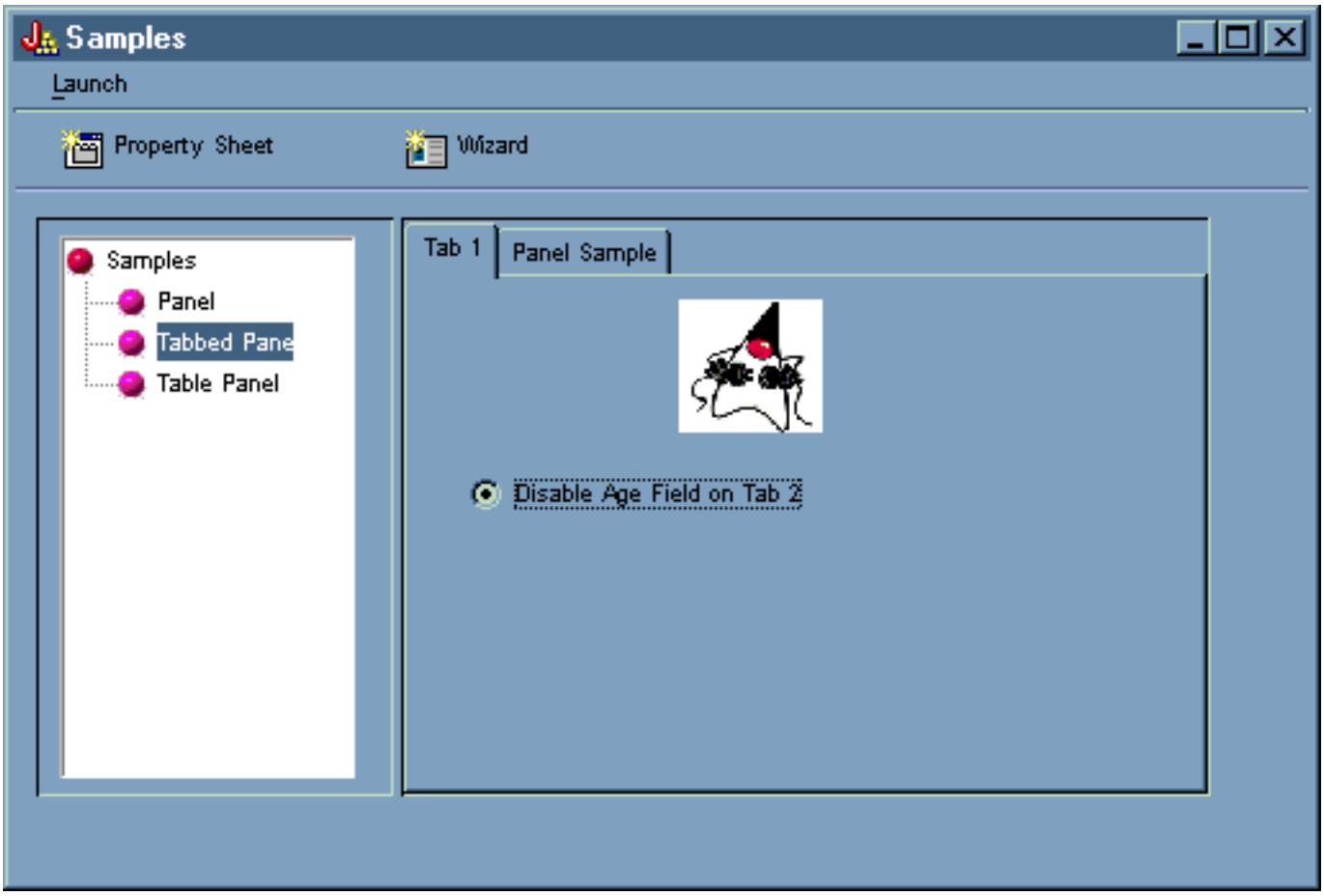
La figura 19 muestra el resultado obtenido de seleccionar la pestaña **Ejemplo de panel** en la sección derecha.

Figura 19: seleccionar la pestaña Ejemplo de panel en la sección derecha



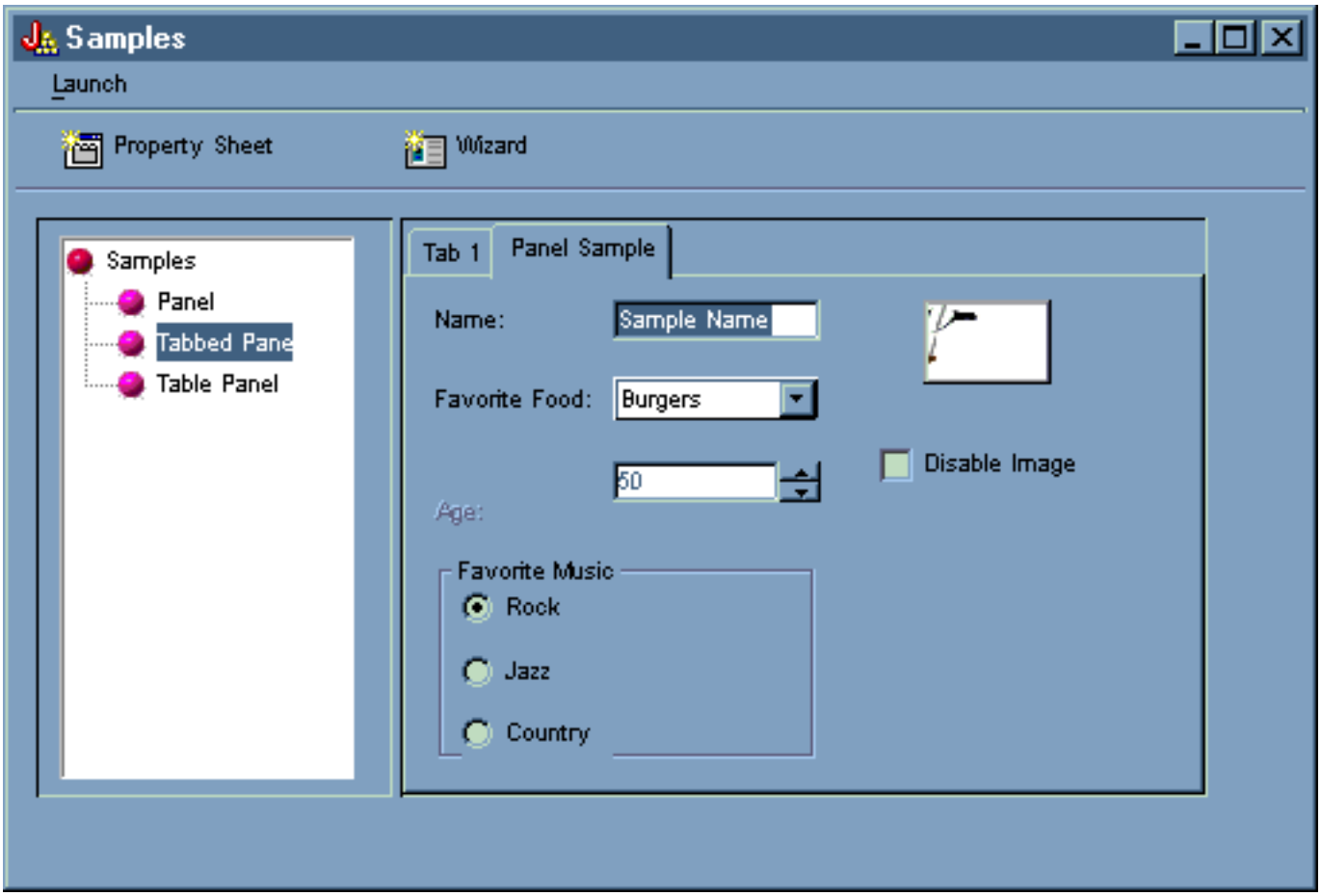
Seleccione **Pestaña 1** de nuevo (en la sección derecha) y, a continuación, pulse **Inhabilitar campo de edad en la pestaña 2** para deseleccionarlo.

Figura 20: seleccionar Inhabilitar campo de edad en la pestaña 2 en la sección derecha



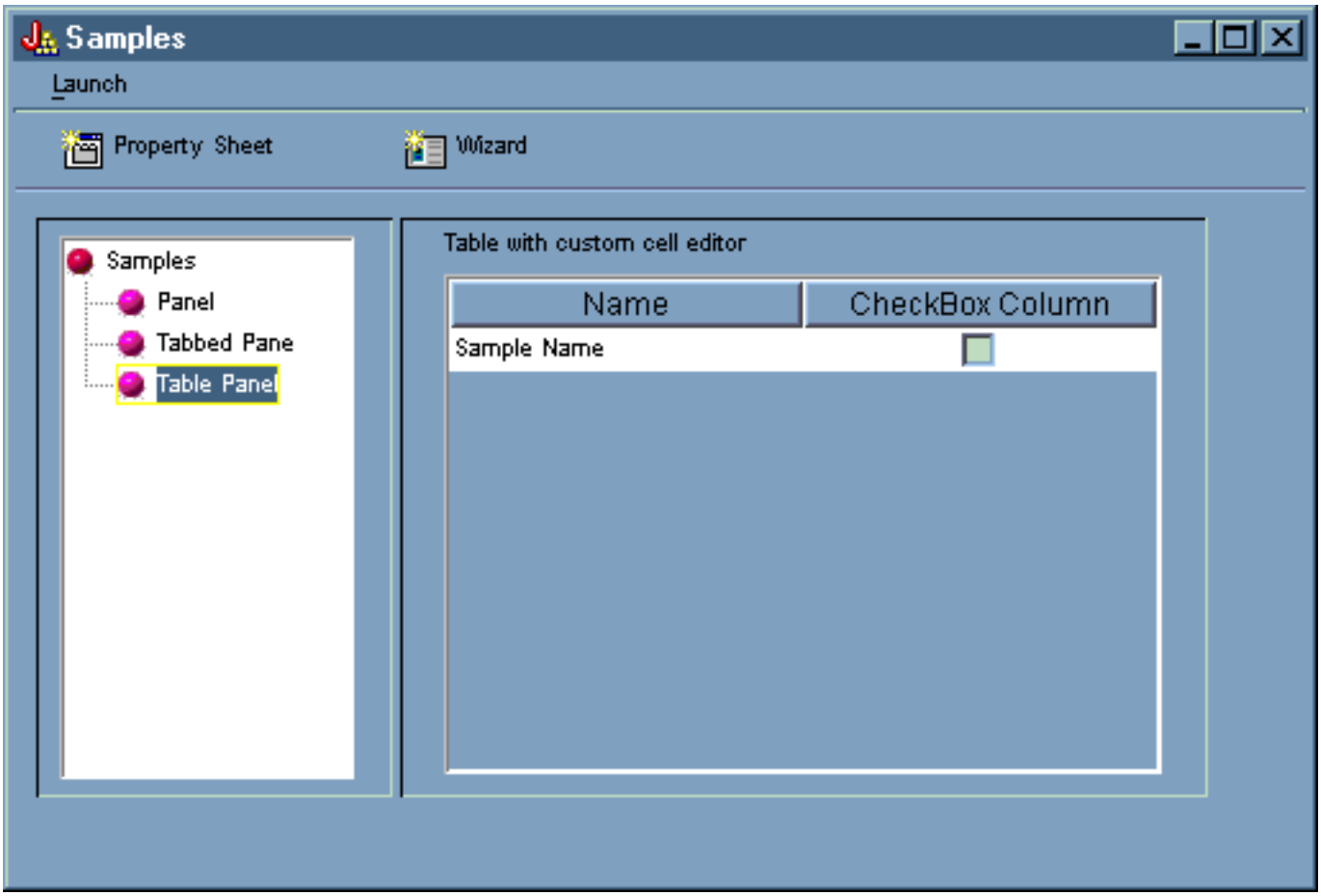
Al seleccionar la opción **Inhabilitar campo de edad en la pestaña 2** se desactiva y se atenúa el campo **Edad** en la pestaña **Ejemplo de panel**, como se muestra en la figura 21.

Figura 21: resultado de inhabilitar la edad en la pestaña Ejemplo de panel



Al seleccionar **Panel de tabla** en la sección izquierda de la ventana principal del ejemplo de Constructor de GUI se muestra el uso de un panel de tabla con un representador personalizado y un editor de casillas personalizado, como se muestra en la figura 22.

Figura 22: seleccionar Panel de tabla en la sección izquierda


Samples
Launch
Property Sheet
Wizard
Samples
Panel
Tabbed Pane
Table Panel
Table with custom cell editor
Name
CheckBox Column
Sample Name

Ejemplos de las clases HTML

Los ejemplos que figuran a continuación muestran algunas maneras de cómo se pueden utilizar las clases HTML:

- Ejemplo: cómo se utiliza la clase BidiOrdering
- Ejemplo: crear objetos HTMLAlign
- Ejemplo: cómo se utilizan las clases de formularios HTML
- Ejemplos de clases de entrada de formulario:
    - Ejemplo: crear un objeto ButtonFormInput
    - Ejemplo: crear un objeto FileFormInput
    - Ejemplo: crear un objeto HiddenFormInput
    - Ejemplo: crear un objeto ImageFormInput
    - Ejemplo: crear un objeto ResetFormInput
    - Ejemplo: crear un objeto SubmitFormInput
    - Ejemplo: crear un objeto TextFormInput
    - Ejemplo: crear un objeto PasswordFormInput
    - Ejemplo: crear un objeto RadioFormInput
    - Ejemplo: crear un objeto CheckboxFormInput
- Ejemplo: crear objetos HTMLHeading
- Ejemplo: cómo se utiliza la clase HTMLHyperlink
- Ejemplo: cómo se utiliza la clase HTMLImage
- Ejemplos de HTMLList
    - Ejemplo: crear listas ordenadas
    - Ejemplo: crear listas sin ordenar
    - Ejemplo: crear listas anidadas
- Ejemplo: crear códigos HTMLMeta
- Ejemplo: crear códigos HTMLParameter
- Ejemplo: crear códigos HTMLServlet
- Ejemplo: cómo se utiliza la clase HTMLText
- Ejemplos de HTMLTree
    - Ejemplo: cómo se utiliza la clase HTMLTree
    - Ejemplo: crear un árbol de sistema de archivos integrado que se pueda recorrer
- Clases de formulario con diseño:
    - Ejemplo: cómo se utiliza la clase GridLayoutFormPanel
    - Ejemplo: cómo se utiliza la clase LineLayoutFormPanel
- Ejemplo: cómo se utiliza la clase TextAreaFormElement
- Ejemplo: cómo se utiliza la clase LabelFormOutput
- Ejemplo: cómo se utiliza la clase SelectFormElement
- Ejemplo: cómo se utiliza la clase SelectOption

- Ejemplo: cómo se utiliza la clase RadioFormInputGroup
- Ejemplo: cómo se utiliza la clase RadioFormInput
- Ejemplo: cómo se utilizan las clases HTMLTable
    - Ejemplo: cómo se utiliza la clase HTMLTableCell
    - Ejemplo: cómo se utiliza la clase HTMLTableRow
    - Ejemplo: cómo se utiliza la clase HTMLTableHeader
    - Ejemplo: cómo se utiliza la clase HTMLTableCaption

También se pueden utilizar juntas las clases HTML y las clases servlet, como en este ejemplo.

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplo: cómo se utilizan las clases de formularios HTML

El ejemplo siguiente muestra cómo se utilizan las clases de formularios HTML. También puede ver una salida de ejemplo a partir de la ejecución de este código. Las clases HTML usadas en el método "showHTML" están en **negrita**.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de cómo se usan las clases del paquete HTML de
IBM
// Toolbox para Java que le permiten construir formularios HTML con
facilidad.
//
///////////////////////////////////////////////////////////////////////////

package customer;

import java.io.*;
import java.awt.Color;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.*;
import com.ibm.as400.util.html.*;


public class HTMLExample extends HttpServlet
{

  private static boolean found = false;    // Determina si el usuario ya
existe en
                                           // la lista de registrados.

  String regPath = "c:\\registration.txt"; // La información de registro se
almacenará aquí.


    public void init(ServletConfig config)
  {

                           try
    {
       super.init(config);

    }
    catch(Exception e)
    {
      e.printStackTrace();
    }

  }

  /**
   *  Procesar la petición GET.
   *  @param req La petición.
   *  @param res La respuesta.
   **/
   public void doGet (HttpServletRequest req, HttpServletResponse res)
       throws ServletException, IOException
   {
       res.setContentType("text/html");
       ServletOutputStream out = res.getOutputStream();

      // Visualizar la Web utilizando las nuevas clases HTML.
      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= "";

        // Corriente de salida para escribir en el servlet.
       ServletOutputStream out = res.getOutputStream();

       res.setContentType("text/html");

        // Comprobar si los parámetros name y e-mail tienen valores válidos.
        if (nameStr.length() == 0)
           errorText += "Nombre de cliente no entrado.  ";
        if (emailStr.length() == 0)
           errorText += "Correo electrónico no entrado.  " ;

        // Si se han proporcionado los parámetros name y e-mail, continuar.
        if (errorText.length() == 0)
        {

                           try
           {
              // Cree el archivo registration.txt.
              FileWriter f = new FileWriter(regPath, true);
              BufferedWriter output = new BufferedWriter(f);

              // Lector puesto en almacenamiento intermedio para buscar en
el archivo.
              BufferedReader in = new BufferedReader(new
FileReader(regPath));

              String line = in.readLine();

              // Restablezca el distintivo de encontrado.
              found = false;

              // Compruebe si este cliente ya se ha registrado
              // o si ya ha utilizado la misma dirección de correo
electrónico.
              while (!found)
              {
                 // Si el archivo está vacío o se ha llegado al final del
archivo.
                 if (line == null)
              break;

                 // Si el cliente ya está registrado.
                 if ((line.equals("Nombre cliente: " + nameStr)) ||
(line.equals("Dirección correo electrónico: " + emailStr)))
                 {
                    // Envíe como salida un mensaje al cliente que diga que
ya
                    // está registrado.
           out.println ("<HTML> " +
                                "<TITLE> Registro de la Caja de
Herramientas</TITLE> " +
                        "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> "
+
                        "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> "
);
                    out.println ("<P><HR>" +
                                "<P>" + nameStr + "</B>, ya está registrado
con ese " +
```

```
                                "<B>Nombre</B> o <B>Dirección de correo
electrónico</B>." +
                                "<P> ¡Gracias!...<P><HR>");

           // Cree un objeto HTMLHyperlink y visualícelo.
                  out.println ("<UL><LI>" + new
HTMLHyperlink("./customer.HTMLExample", "De nuevo en el formulario de
registro") + "</UL></BODY></HTML>");
                  found = true;
              break;

                 }
                 else    // Lea la línea siguiente.
                    line = in.readLine();

              }

              // Objeto String para contener los datos sometidos desde el
formulario HTML.
              String data;

              // Si en nuestro archivo de texto no se ha encontrado el
nombre o el correo electrónico del usuario, continuar.
              if (!found)
              {

//---------------------------------------------------------------------------
                 // Inserte la nueva información de cliente en un archivo.
                       output.newLine();
                 output.write("Nombre de cliente: " + nameStr);
                       output.newLine();
                 output.write("Dirección de correo electrónico: " +
emailStr);
                       output.newLine();

//---------------------------------------------------------------------------



//---------------------------------------------------------------------------
                 // Obtener el recuadro de selección "USE" a partir del
formulario.
                 data = req.getParameter("use");
                 if(data != null)
                 {
                    output.write("Utiliza actualmente la Caja de
Herramientas: " + data);
                       output.newLine();
                 }

//---------------------------------------------------------------------------



//---------------------------------------------------------------------------
                 // Obtener el recuadro de selección "Más información" a
partir del formulario.
                 data = req.getParameter("contact");
         if (data != null)
                 {
                    output.write("Ha solicitado más información: " + data);
                       output.newLine();
                 }

//---------------------------------------------------------------------------



//---------------------------------------------------------------------------
```

```
                // Obtener "Versión de AS400" a partir del formulario.
                data = req.getParameter("version");
        if (data != null)
                {
                   if (data.equals("multiple versions"))
                   {
                      data = req.getParameter("MultiList");
                      output.write("Múltiples versiones: " + data);
                   }
          else
                      output.write("Versión de AS400: " + data);

                      output.newLine();
                }

//-----------------------------------------------------------------------------



//-----------------------------------------------------------------------------
                // Obtener "Proyectos actuales" a partir del formulario.
                data = req.getParameter("interest");
        if (data != null)
                {
                   output.write("Utiliza Java o está interesado en: " +
data);
                      output.newLine();
                }

//-----------------------------------------------------------------------------



//-----------------------------------------------------------------------------
                // Obtener "Plataformas" a partir del formulario.
                data = req.getParameter("platform");
        if (data != null)
                {
                   output.write("Plataformas: " + data);
                      output.newLine();
                   if (data.indexOf("Other") >= 0)
                   {
                      output.write("Otras plataformas: " +
req.getParameter("OtherPlatforms"));
                      output.newLine();
                   }
                }

//-----------------------------------------------------------------------------



//-----------------------------------------------------------------------------
                // Obtenga "Número de servidores iSeries o AS/400e" del
formulario.
                data = req.getParameter("list1");
        if (data != null)
                {
                   output.write("Número de servidores iSeries: " + data);
                      output.newLine();
                }

//-----------------------------------------------------------------------------



//-----------------------------------------------------------------------------
                // Obtener "Comentarios" a partir del formulario.
                data = req.getParameter("comments");
```

```
                if (data != null && data.length() > 0)
                {
                   output.write("Comentarios: " + data);
                      output.newLine();
                }

//-------------------------------------------------------------------------------



//-------------------------------------------------------------------------------
                // Obtener el archivo "adjunto".
                data = req.getParameter("myAttachment");
                if (data != null && data.length() > 0)
                {
                   output.write("Archivo adjunto: " + data);
                      output.newLine();
                }

//-------------------------------------------------------------------------------



//-------------------------------------------------------------------------------
                // Obtener información de "Copyright" oculta.
                data = req.getParameter("copyright");
        if (data != null)
                {
                   output.write(data);
                      output.newLine();
                }

//-------------------------------------------------------------------------------


                output.flush();
                output.close();

                // Imprimir un agradecimiento al cliente.
              out.println("<HTML>");
                out.println("<TITLE>¡Gracias!</TITLE>");
              out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\">
");
              out.println("<BODY BGCOLOR=\"blanchedalmond\">");
                out.println("<HR><P>Gracias por haberse registrado, <B>" +
nameStr + "</B>!<P><HR>");

            // Cree un objeto HTMLHyperlink y visualícelo.
              out.println("<UL><LI>" + new
HTMLHyperlink("./customer.HTMLExample", "De nuevo en el formulario de
registro"));
              out.println("</UL></BODY></HTML>");

              }

             }
        catch (Exception e)
           {
              // Mostrar error en el navegador.
              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>Mensaje de error:</B><P>");
              out.println(e + "<P>");

            // Cree un objeto HTMLHyperlink y visualícelo.
              out.println("<UL><LI>" + new
```

```
HTMLHyperlink("./customer.HTMLExample", "De nuevo en el formulario de
registro"));
              out.println("</UL></BODY></HTML>");

      e.printStackTrace();
           }
        }
          else
        {
           // Envíe como salida un mensaje al cliente que diga que no ha
entrado el nombre ni
           // la dirección de correo electrónico y que lo intente otra vez.
           out.println ("<HTML> " +
                        "<TITLE>Formulario de registro no válido</TITLE> " +
                        "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> "
+
                        "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> "
);

           out.println ("<HR><B>ERROR</B> en los datos de cliente - <P><B>"
+
                        errorText + "</B><P> Inténtelo otra vez, por
favor... <HR>");


           // Cree un objeto HTMLHyperlink y visualícelo.
           out.println ("<UL><LI>" + new
HTMLHyperlink("./customer.HTMLExample", "De nuevo en el formulario de
registro") + "</UL></BODY></HTML>");
        }
         // Cierre el transcriptor.
         out.close();


    }



    public void destroy(ServletConfig config)
    {
       // No haga nada.
    }



    public String getServletInfo()
    {
        return "Registro de mi producto";
    }

    private String showHTML()
    {
      // Almacenamiento intermedio de tipo String para contener la página
HTML.
    StringBuffer page = new StringBuffer();

      // Cree el objeto formulario HTML.
      HTMLForm form = new HTMLForm("/servlet/customer.HTMLExample");;
      HTMLText txt;

      // Construya el principio de la página HTML y añádalo al
almacenamiento intermedio de tipo String.
      page.append("<HTML>\n");
      page.append("<TITLE> ¡Bienvenido!</TITLE>\n");
      page.append("<HEAD><SCRIPT LANGUAGE=\"JavaScript\">function
test(){alert(\"Esto es un script de ejemplo ejecutado con un
ButtonFormInput.\")}</SCRIPT></HEAD>");
      page.append("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\">\n");
```

```
      page.append("<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"><BR>\n");

                              try
      {

//-------------------------------------------------------------------------------------------
         // Cree un título de página con texto HTML.
        txt = new HTMLText("Registro de producto");
        txt.setSize(5);
         txt.setBold(true);
         txt.setColor(new Color(199, 21, 133));
         txt.setAlignment(HTMLConstants.CENTER);

         // Añada texto HTML al almacenamiento intermedio de tipo String.
         page.append(txt.getTag(true) + "<HR><BR>\n");

//-------------------------------------------------------------------------------------------


//-------------------------------------------------------------------------------------------
         // Cree un diseño lineal.
         LineLayoutFormPanel line = new LineLayoutFormPanel();
         txt = new HTMLText("Entre su nombre y dirección de correo
electrónico:");
         txt.setSize(4);
         line.addElement(txt);

         // Añada el diseño lineal al almacenamiento intermedio de tipo
String.
         page.append(line.toString());
         page.append("<BR>");

//-------------------------------------------------------------------------------------------


//-------------------------------------------------------------------------------------------
         // Establezca el METHOD del formulario HTML.
        form.setMethod(HTMLForm.METHOD_POST);

//-------------------------------------------------------------------------------------------


//-------------------------------------------------------------------------------------------
         // Cree una entrada de texto para el nombre.
         TextFormInput user = new TextFormInput("name");
         user.setSize(25);
         user.setMaxLength(40);

         // Cree una entrada de texto para la dirección de correo-e.
         TextFormInput email = new TextFormInput("email");
         email.setSize(30);
         email.setMaxLength(40);

         // Cree una ImageFormInput.
         ImageFormInput img = new ImageFormInput("Someter formulario",
"..\\images\\myPiimages/c.gif");
         img.setAlignment(HTMLConstants.RIGHT);

//-------------------------------------------------------------------------------------------


//-------------------------------------------------------------------------------------------
         // Cree un objeto LineLayoutFormPanel para el nombre y la dirección
de correo-e.
         LineLayoutFormPanel line2 = new LineLayoutFormPanel();

         // Añada elementos al formulario lineal.
         line2.addElement(new LabelFormElement("Nombre:"));
         line2.addElement(user);
```

```
        // Cree y añade un elemento etiqueta al diseño lineal.
        line2.addElement(new LabelFormElement("Correo electrónico:"));
        line2.addElement(email);
        line2.addElement(img);

//-----------------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------------
        // Cree un diseño lineal para hacer preguntas.
        LineLayoutFormPanel line3 = new LineLayoutFormPanel();

        // Añada elementos al diseño lineal.
        line3.addElement(new LineLayoutFormPanel());
        line3.addElement(new CheckboxFormInput("use", "yes", "¿Utiliza
actualmente la Caja de Herramientas?", false));
        line3.addElement(new LineLayoutFormPanel());
        line3.addElement(new CheckboxFormInput("contact", "yes", "¿Desea
información sobre los futuros releases de la Caja de Herramientas?", true));
        line3.addElement(new LineLayoutFormPanel());

//-----------------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------------
        // Cree un grupo de botones de selección de versión.
        RadioFormInputGroup group = new RadioFormInputGroup("version");

        // Añada al grupo entradas de formulario que sean botones de
selección.
        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",
"¿Múltiples versiones? Cuáles:", false));

        // Cree un elemento de formulario de selección.
        SelectFormElement mlist = new SelectFormElement("MultiList");
        mlist.setMultiple(true);
        mlist.setSize(3);

        // Cree las opciones para el elemento de formulario de selección.
        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");

        // Cree texto HTML.
        txt = new HTMLText("Nivel actual del servidor:");
        txt.setSize(4);

        // Cree un diseño cuadriculado.
        GridLayoutFormPanel grid1 = new GridLayoutFormPanel(3);

        // Añada el grupo de botones de selección y el elemento de
formulario de selección a la cuadrícula.
        grid1.addElement(txt);
        grid1.addElement(group);
        grid1.addElement(mlist);

//-----------------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------------
        // Cree un diseño cuadricualdo para los intereses.
        GridLayoutFormPanel grid2 = new GridLayoutFormPanel(1);
```

```
        txt = new HTMLText("Proyectos actuales o área de interés: (marque
todo lo que crea oportuno)");
        txt.setSize(4);

        // Añada elementos al diseño cuadriculado.
        grid2.addElement(new LineLayoutFormPanel());
        grid2.addElement(txt);
        // Cree y añada un recuadro de selección al diseño cuadriculado.
        grid2.addElement(new CheckboxFormInput("interest", "applications",
"Aplicaciones", true));
        grid2.addElement(new CheckboxFormInput("interest", "applets",
"Applets", false));
        grid2.addElement(new CheckboxFormInput("interest", "servlets",
"Servlets", false));

//-------------------------------------------------------------------------------------


//-------------------------------------------------------------------------------------
        // Cree un diseño lineal para las plataformas.
        LineLayoutFormPanel line4 = new LineLayoutFormPanel();
        txt = new HTMLText("Plataformas de cliente utilizadas: (marque todo
lo que crea oportuno)");
        txt.setSize(4);

        // Añada elementos al diseño lineal.
        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",
"Otras:", false));

        TextFormInput other = new TextFormInput("OtherPlatforms");
        other.setSize(20);
        other.setMaxLength(50);

        line4.addElement(other);

//-------------------------------------------------------------------------------------


//-------------------------------------------------------------------------------------
        // Cree un diseño lineal para el número de servidores.
        LineLayoutFormPanel grid3 = new LineLayoutFormPanel();

        txt = new HTMLText("¿Cuántos servidores iSeries o AS/400e tiene?
");
        txt.setSize(4);

        // Cree un elemento de formulario de selección para el número de
servidores poseídos.
  SelectFormElement list = new SelectFormElement("list1");
        // Cree y añada las opciones de selección a la lista de elementos
de formulario de selección.
        SelectOption opt0 = list.addOption("0", "cero");
```

```
        SelectOption opt1 = list.addOption("1", "uno", true);
        SelectOption opt2 = list.addOption("2", "dos");
        SelectOption opt3 = list.addOption("3", "tres");
        SelectOption opt4 = list.addOption("4", "cuatro");
        SelectOption opt5 = new SelectOption("5+", "CincoOMás", false);
        list.addOption(opt5);

        // Añada elementos al diseño cuadriculado.
        grid3.addElement(new LineLayoutFormPanel());
        grid3.addElement(txt);
        grid3.addElement(list);

//-----------------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------------
        // Cree un diseño cuadriculado para los comentarios de producto.
        GridLayoutFormPanel grid4 = new GridLayoutFormPanel(1);
        txt = new HTMLText("Comentarios de producto:");
        txt.setSize(4);

        // Añada elementos al diseño cuadriculado.
        grid4.addElement(new LineLayoutFormPanel());
        grid4.addElement(txt);
        //grid4.addElement(new LineLayoutFormPanel());
        // Cree un formulario de área de texto.
        grid4.addElement(new TextAreaFormElement("comments", 5, 75));
        grid4.addElement(new LineLayoutFormPanel());

//-----------------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------------
        // Cree un diseño cuadriculado.
        GridLayoutFormPanel grid5 = new GridLayoutFormPanel(2);
        txt = new HTMLText("¿Desea iniciar la sesión en un servidor?");
        txt.setSize(4);

        // Cree una entrada de texto y una etiqueta para el nombre del
sistema.
        TextFormInput sys = new TextFormInput("system");
        LabelFormElement sysLabel = new LabelFormElement("Sistema:");

        // Cree una entrada de texto y una etiqueta para el ID de usuario.
        TextFormInput uid = new TextFormInput("uid");
        LabelFormElement uidLabel = new LabelFormElement("ID de usuario");

        // Cree una entrada de contraseña y una etiqueta para la
contraseña.
        PasswordFormInput pwd = new PasswordFormInput("pwd");
        LabelFormElement pwdLabel = new LabelFormElement("Contraseña");

        // Añada las entradas de texto, las entradas de contraseña y las
etiquetas a la cuadrícula.
        grid5.addElement(sysLabel);
        grid5.addElement(sys);
        grid5.addElement(uidLabel);
        grid5.addElement(uid);
        grid5.addElement(pwdLabel);
        grid5.addElement(pwd);

//-----------------------------------------------------------------------------------------



//-----------------------------------------------------------------------------------------
        // Añada los diversos paneles creados al formulario HTML en el
orden en que han de aparecer.
        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("Someter un archivo adjunto aquí: <br
/>"));
          // Añada una entrada de archivo al formulario
          form.addElement(new FileFormInput("myAttachment"));
          form.addElement(new ButtonFormInput("button", "¡PULSE AQUÍ!",
"test()"));
          // Añada un diseño lineal vacío que a su vez
          // añada una división de línea <br /> al formulario.
          form.addElement(new LineLayoutFormPanel());
          form.addElement(new LineLayoutFormPanel());
          form.addElement(new SubmitFormInput("submit", "Registrar"));
          form.addElement(new LineLayoutFormPanel());
          form.addElement(new LineLayoutFormPanel());
          form.addElement(new ResetFormInput("reset", "Restablecer"));

          // Añada una entrada oculta al formulario.
          form.addElement(new HiddenFormInput("copyright", "(C) Copyright IBM
Corp. 1999, 1999"));

//-----------------------------------------------------------------------------------------

          // Añada todo el formulario HTML al almacenamiento intermedio de
tipo String.
      page.append(form.toString());

      }
      catch(Exception e)
      {
      e.printStackTrace();
      }

      // Añada los códigos HTML de final al almacenamiento intermedio.
      page.append("</BODY>\n");
      page.append("</HTML>\n");

      // Devuelva toda la página HTML de tipo serie.
    return page.toString();

   }
}
```

Ejemplo: cómo se utilizan las clases HTMLTree

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de cómo se usan las clases del paquete HTML de
IBM
// Toolbox para Java que permiten construir fácilmente árboles de archivo y
HTML.
//
///////////////////////////////////////////////////////////////////////////

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;

/**
 * Ejemplo de cómo se usan las clases HTMLTree y FileTreeElement en un
servlet.
 **/
public class TreeNav extends HttpServlet
{
    public void init(ServletConfig config)
       throws ServletException
   {
       super.init(config);

      // La Caja de Herramientas emplea un conjunto de iconos por omisión
para representar los elementos expandidos,
      // contraídos y documentos dentro de HTMLTree. Para mejorarlos,
proporciona tres archivos .gif (expanded.gif,
      // collapsed.gif, bullet.gif) en el archivo jt400Servlet.jar. Los
navegadores no encuentran archivos .gif en un
      // archivo .jar o .zip, por lo que deben extraerse esas imágenes del
archivo .jar y ponerse en el directorio de
      // servidor Web adecuado (por omisión es el directorio /html). Después
descomente las siguientes líneas de código
      // y especifique la ubicación correcta en estos métodos set. La
ubicación puede ser absoluta o relativa.

      HTMLTreeElement.setExpandedGif("/images/expanded.gif");
```

```
      HTMLTreeElement.setCollapsedGif("/images/collapsed.gif");
      HTMLTreeElement.setDocGif("/images/bullet.gif");
   }

   /**
   *  Procesar la petición GET.
   *  @param req La petición.
   *  @param res La respuesta.
    **/
   public void doGet (HttpServletRequest req, HttpServletResponse resp)
       throws ServletException, IOException
   {
      HttpSession session = req.getSession(true);
      HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

      // Si esta sesión todavía no tiene ningún árbol de archivo,
      // cree el árbol inicial.
      if (fileTree == null)
         fileTree = createTree(req, resp, req.getRequestURI());

      // Establezca la petición de servlet HTTP en 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");

      // Obtenga el código de HTMLTree.
      out.println(fileTree.getTag());

      out.println("</body>\n");
    out.println("</html>\n");
         out.close();

      // Establezca el valor de árbol de la sesión, para que al acceder a
      // este servlet por segunda vez, se reutilice el objeto FileTree.
    session.putValue("filetree", fileTree);
   }

   /**
   *  Procesar la petición POST.
   *  @param req La petición.
   *  @param res La respuesta.
    **/
    public void doPost (HttpServletRequest req, HttpServletResponse res)
       throws ServletException, IOException
   {
      res.setContentType("text/html");
       ServletOutputStream out = res.getOutputStream();
   }


   /**
    *  Este método creará el objeto HTMLTree inicial.
    **/
   private HTMLTree createTree(HttpServletRequest req, HttpServletResponse
resp, String uri)
   {
      // Cree un objeto HTMLTree.
```

```
      HTMLTree tree = new HTMLTree(req);

                              try
    {
          // Cree un objeto URLParser.
          URLParser urlParser = new URLParser(uri);

         AS400 sys = new AS400(CPUStatus.systemName_, "javactl", "jteam1");

          // Cree un objeto Archivo y establezca el directorio de IFS raíz.
       IFSJavaFile root = new IFSJavaFile(sys, "/QIBM");

         // Cree un filtro y liste todos los directorios.
          DirFilter filter = new DirFilter();
         //File[] dirList = root.listFiles(filter);

          // Obtenga la lista de archivos que se adecuan al filtro de
directorios.
          String[] list = root.list(filter);

          File[] dirList = new File[list.length];

          // No queremos obligar a los servidores Web a utilizar JDK1.2 ya
que la
         // mayoría de las JVM de servidor Web son más lentas de actualizar
al
         // último nivel de JDK. Lo más eficaz para crear estos objetos de
archivo
         // es usar el método listFiles(filter) en JDK1.2, lo que se haría
como
         // se indica a continuación, en lugar de utilizar el método
list(filter)
         // y después convertir la matriz de serie devuelta en la matriz de
         // archivo adecuada.
          // 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++)
       {
            // Cree un objeto FileTreeElement para cada directorio de la
lista.
            FileTreeElement node = new FileTreeElement(dirList[i]);

            // Cree un objeto ServletHyperlink para los iconos de
expandir/contraer.
            ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
            //sl.setHttpServletResponse(resp);
            node.setIconUrl(sl);


            // Cree un objeto ServletHyperlink para el servlet TreeList, que
            // visualizará el contenido del objeto FileTreeElement
(directorio).
        ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
```

```
        tl.setTarget("list");

            // Si ServletHyperlink no tiene nombre, establézcalo en el
            // nombre del directorio.
            if (tl.getText() == null)
              tl.setText(dirList[i].getName());

            // Establezca TextUrl para el objeto FileTreeElement.
            node.setTextUrl(tl);

            // Añada el objeto FileTreeElement a HTMLTree.
            tree.addElement(node);
        }
     }
        catch (Exception e)
     {
       e.printStackTrace();
     }

        return tree;
   }


    public void destroy(ServletConfig config)
    {
       // No haga nada.
    }

    public String getServletInfo()
    {
        return "Navegación por FileTree";
    }
}
```

Ejemplo: crear un árbol de sistema de archivos integrado que se pueda recorrer (Archivo 1 de 3)

Este código de ejemplo, junto con el código de los dos archivos de ejemplo adicionales, visualiza un objeto HTMLTree y un objeto FileListElement en un servlet. Los tres archivos del ejemplo son:

- FileTreeExample.java - Este archivo genera los marcos HTML y arranca el servlet.
- TreeNav.java - Este archivo construye y gestiona el árbol.
- TreeList.java - Este archivo visualiza el contenido de las selecciones efectuadas en la clase TreeNav.java.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de cómo se usan las clases del paquete HTML de
IBM
// Toolbox para Java que permiten construir fácilmente árboles de archivo y
HTML.
//
///////////////////////////////////////////////////////////////////////////

import java.io.PrintWriter;
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.HTMLMeta;


//
// Ejemplo de cómo se utilizan los marcos para visualizar un HTMLTree y un
FileListElement
// en un servlet.
//

public class FileTreeExample extends HttpServlet
{
    public void init(ServletConfig config)
       throws ServletException
   {
       super.init(config);
   }

   /**
   *  Procesar la petición GET.
   *  @param req La petición.
   *  @param res La respuesta.
    **/

   public void doGet (HttpServletRequest req, HttpServletResponse resp)
       throws ServletException, IOException
   {
      resp.setContentType("text/html");
```

```
      // Defina dos marcos. El primero, un marco de navegación, visualizará
      // el HTMLTree, que contendrá FileTreeElements y hará posible la
      // navegación del sistema de archivos. El segundo visualizará/listará
      // el contenido de un directorio seleccionado en el marco de
navegación.
         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();
   }

   /**
   *  Procesar la petición POST.
   *  @param req La petición.
   *  @param res La respuesta.
    **/

    public void doPost (HttpServletRequest req, HttpServletResponse res)
       throws ServletException, IOException
   {
       res.setContentType("text/html");
       ServletOutputStream out = res.getOutputStream();
   }

    public void destroy(ServletConfig config)
   {
       // No haga nada.
   }

    public String getServletInfo()
   {
      return "Servlet FileTree";
   }
}
```

Ejemplo: crear un árbol de sistema de archivos integrado que se pueda recorrer (Archivo 2 de 3)

Este código de ejemplo, junto con el código de los dos archivos de ejemplo adicionales, visualiza un objeto HTMLTree y un objeto FileListElement en un servlet. Los tres archivos del ejemplo son:

- FileTreeExample.java - Este archivo genera los marcos HTML y arranca el servlet.
- TreeNav.java - Este archivo construye y gestiona el árbol.
- TreeList.java - Este archivo visualiza el contenido de las selecciones efectuadas en la clase TreeNav.java.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de cómo se usan las clases del paquete HTML de
IBM
// Toolbox para Java que permiten construir fácilmente árboles de archivo y
HTML.
//
////////////////////////////////////////////////////////////////////////////

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;

//
// Ejemplo de cómo se usan las clases HTMLTree y FileTreeElement
// en un servlet.
//

public class TreeNav extends HttpServlet
{
   private AS400 sys_;

    public void init(ServletConfig config)
       throws ServletException
   {
       super.init(config);

       // Cree un objeto AS400.
      sys_ = new AS400("mySystem", "myUserID", "myPassword");
```

```
      // La Caja de Herramientas emplea un conjunto de iconos por omisión
para representar elementos expandidos,
      // contraídos y documentos dentro del objeto HTMLTree. Para mejorar
estos iconos, proporciona
      // tres archivos .gif (expanded.gif, collapsed.gif y bullet.gif) en el
archivo jt400Servlet.jar.
      // Los navegadores no encuentran archivos .gif en un archivo jar o
zip, por lo que
      // es necesario extraer estas imágenes del archivo jar y colocarlas en
el directorio
      // de servidor Web adecuado (por omisión es el directorio /html). A
continuación,
      // cambie las siguientes líneas de código para especificar la
ubicación correcta en los
      // métodos set. La ubicación puede ser absoluta o relativa.

      HTMLTreeElement.setExpandedGif("http://myServer/expanded.gif");
      HTMLTreeElement.setCollapsedGif("http://myServer/collapsed.gif");
      HTMLTreeElement.setDocGif("http://myServer/bullet.gif");
   }

   /**
   *  Procesar la petición GET.
   *  @param req La petición.
   *  @param res La respuesta.
    **/

   public void doGet (HttpServletRequest req, HttpServletResponse resp)
       throws ServletException, IOException
   {
      // Utilice los datos de sesión para recordar el estado del árbol.
      HttpSession session = req.getSession(true);
      HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

      // Si esta sesión todavía no tiene ningún árbol de archivo,
      // cree el árbol inicial.
      if (fileTree == null)
         fileTree = createTree(req, resp, req.getRequestURI());

      // Establezca la petición de servlet HTTP en 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");

      // Obtenga el código de HTMLTree.
      out.println(fileTree.getTag());

         out.println("</body>\n");
         out.println("</html>\n");
         out.close();

      // Establezca el valor de árbol de la sesión, para que al acceder a
      // este servlet por segunda vez, se reutilice el objeto FileTree.
      session.putValue("filetree", fileTree);
   }

   /**
```

```
   *  Procesar la petición POST.
   *  @param req La petición.
   *  @param res La respuesta.
    **/

    public void doPost (HttpServletRequest req, HttpServletResponse res)
       throws ServletException, IOException
   {
      res.setContentType("text/html");
       ServletOutputStream out = res.getOutputStream();
   }

   /**
    *  Este método creará el objeto HTMLTree inicial.
    **/

   private HTMLTree createTree(HttpServletRequest req,
                               HttpServletResponse resp, String uri)
   {
      // Cree un objeto HTMLTree.
      HTMLTree tree = new HTMLTree(req);

                            try
        {
       // Cree un objeto URLParser.
          URLParser urlParser = new URLParser(uri);

          // Cree un objeto Archivo y establezca el directorio de IFS raíz.
          IFSJavaFile root = new IFSJavaFile(sys_, "/QIBM");

          // Cree un filtro.
          DirFilter filter = new DirFilter();

          // Obtenga la lista de archivos que se adecuan al filtro de
directorios.
          String[] list = root.list(filter);

          File[] dirList = new File[list.length];

          // No queremos obligar a los servidores Web a utilizar JDK1.2 ya
que la
          // mayoría de las JVM de servidor Web son más lentas de actualizar
al
          // último nivel de JDK. Lo más eficaz para crear estos objetos de
archivo
          // es usar el método listFiles(filter) en JDK1.2, lo que se haría
como
          // se indica a continuación, en lugar de utilizar el método
list(filter)
          // y después convertir la matriz de serie devuelta en la matriz de
          // matriz de archivo adecuada.
          // 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++)
```

```
          {
            // Cree un objeto FileTreeElement para cada directorio de la
lista.
            FileTreeElement node = new FileTreeElement(dirList[i]);

            // Cree un objeto ServletHyperlink para los iconos de
expandir/contraer.
            ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
            sl.setHttpServletResponse(resp);
            node.setIconUrl(sl);

            // Cree un objeto ServletHyperlink para el servlet TreeList, que
            // visualizará el contenido del objeto FileTreeElement
(directorio).
            ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
            tl.setTarget("list");

            // Si ServletHyperlink no tiene nombre, establézcalo en el
            // nombre del directorio.
            if (tl.getText() == null)
              tl.setText(dirList[i].getName());

            // Establezca TextUrl para el objeto FileTreeElement.
            node.setTextUrl(tl);

            // Añada el objeto FileTreeElement a HTMLTree.
            tree.addElement(node);
          }

          sys_.disconnectAllServices();
        }

        catch (Exception e)
        {
      e.printStackTrace();
        }

        return tree;
    }

    public void destroy(ServletConfig config)
    {
       // No haga nada.
    }

    public String getServletInfo()
    {
        return "Navegación por FileTree";
    }
}
```

Ejemplo: crear un árbol de sistema de archivos integrado que se pueda recorrer (Archivo 3 de 3)

Este código de ejemplo, junto con el código de los dos archivos de ejemplo adicionales, visualiza un objeto HTMLTree y un objeto FileListElement en un servlet. Los tres archivos del ejemplo son:

- FileTreeExample.java - Este archivo genera los marcos HTML y arranca el servlet.
- TreeNav.java - Este archivo construye y gestiona el árbol.
- TreeList.java - Este archivo visualiza el contenido de las selecciones efectuadas en la clase TreeNav.java.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Este fuente es un ejemplo de cómo se usan las clases del paquete HTML de
IBM
// Toolbox para Java que permiten construir fácilmente listas de archivos y
HTML.
//
///////////////////////////////////////////////////////////////////////////

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.*;

/**
 * Ejemplo de cómo se utiliza la clase FileListElement en un servlet.
 **/
public class TreeList extends HttpServlet
{

   private AS400 sys_;

    /**
   *  Procesar la petición GET.
   *  @param req La petición.
   *  @param res La respuesta.
     **/
    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");

            // Si el parámetro de vía de acceso no es nulo, el usuario ha
seleccionado
            // un elemento en la lista FileTreeElement del marco de
navegación.
         if (req.getPathInfo() != null)
            {
                // Cree un objeto FileListElement que pase un objeto de
sistema AS400 y la petición de servlet HTTP.
                // La petición contendrá la información de vía de acceso
necesaria para listar el contenido del
                // objeto FileTreeElement (directorio) seleccionado.
            FileListElement fileList = new FileListElement(sys_, req);

                //  También puede crear un objeto FileListElement a partir
de un nombre de compartimiento NetServer y una vía de acceso de
compartimiento.
                //
                //  FileListElement fileList = new FileListElement(sys_,
req, "TreeShare", "/QIBM/ProdData/HTTP/Public/jt400");


            // Visualice el contenido de FileListElement.
            out.println(fileList.list());
            }
         else  // Visualice este HTMLHeading si no se ha seleccionado ningún
objeto FileTreeElement.
            {
                HTMLHeading heading = new HTMLHeading(1,"Ejemplo de lista de
archivos HTML");
            heading.setAlign(HTMLConstants.CENTER);

            out.println(heading.getTag());
            }

            out.println("</body>\n");
            out.println("</html>\n");
         out.close();
        }
        catch (Exception e)
        {
      e.printStackTrace();
        }

    }

    /**
   *  Procesar la petición POST.
   *  @param req La petición.
   *  @param res La respuesta.
    **/
```

```
    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);

       // Cree un objeto AS400.
        sys_ = new AS400("mySystem", "myUID", "myPWD");
    }
}
```

Ejemplo: cómo se utilizan las clases HTMLTable

El ejemplo que figura a continuación muestra cómo funcionan las clases HTMLTable:

```
// Cree un objeto HTMLTable por omisión.
HTMLTable table = new HTMLTable();

// Establezca los atributos de la tabla.
table.setAlignment(HTMLTable.CENTER);
table.setBorderWidth(1);

// Cree un objeto HTMLTableCaption por omisión y establezca el texto del pie
de tabla.
HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Saldos de cuenta de cliente - 1 de enero de 2000");

// Establezca el pie de tabla.
table.setCaption(caption);

// Cree las cabeceras de la tabla y añádalas a la misma.
HTMLTableHeader account_header = new HTMLTableHeader(new
HTMLText("CUENTA"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("NOMBRE"));
HTMLTableHeader balance_header = new HTMLTableHeader(new HTMLText("SALDO"));

table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);

// Añada filas a la tabla. Cada registro de cliente corresponde a una fila
de la tabla.
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));

   // Añada la fila a la tabla.
   table.addRow(row);
}
System.out.println(table.getTag());
```

El ejemplo de código Java anterior genera el siguiente código HTML:

```
<table align="center" border="1">
<caption>Saldos de cuenta de cliente - 1 de enero de 2000</caption>
<tr>
<th>CUENTA</th>
<th>NOMBRE</th>
<th>SALDO</th>
```

```
</tr>
<tr align="center">
<td>0000001</td>
<td>Cliente1</td>
<td>100,00</td>
</tr>
<tr align="center">
<td>0000002</td>
<td>Cliente2</td>
<td>200,00</td>
</tr>
<tr align="center">
<td>0000003</td>
<td>Cliente3</td>
<td>550,00</td>
</tr>

</table>
```

La tabla que hay a continuación muestra cómo se visualiza el código HTML anterior en un navegador Web.

Saldos de cuenta de cliente - 1 de enero de 2000

CUENTA	NOMBRE	SALDO
0000001	Cliente1	100,00
0000002	Cliente2	200,00
0000003	Cliente3	550,00

Ejemplos: PCML (Program Call Markup Language)

Los ejemplos siguientes utilizan PCML para llamar a interfaces API de OS/400 y cada uno enlaza con una página que muestra el fuente PCML seguido de un programa Java.

- Ejemplo simple de recuperación de datos: muestra el fuente PCML y el programa Java necesarios para recuperar información acerca de un perfil de usuario en el servidor. En este ejemplo se llama a la API *Recuperar información de usuario* (**QSYRUSRI**).
- Recuperar una lista de información: muestra el fuente PCML y el programa Java necesarios para recuperar una lista de usuarios autorizados en un servidor. En este ejemplo se llama a la API *Abrir lista de usuarios autorizados* (**QGYOLAUS**). Este ejemplo muestra cómo se accede a una matriz de estructuras devuelta por un programa del servidor.
- Recuperar datos multidimensionales: muestra el fuente PCML y el programa Java necesarios para recuperar una lista de exportaciones NFS (sistema de archivos de red) de un servidor. En este ejemplo se llama a la API *Recuperar exportaciones NFS* (**QZNFRTVE**). Este ejemplo muestra cómo se accede a matrices de estructuras dentro de una matriz de estructuras.

Nota: la autorización adecuada varía en cada uno de los ejemplos pero puede incluir determinadas autorizaciones sobre objeto y autorizaciones especiales. Para ejecutar estos ejemplos, debe iniciar la sesión con un perfil de usuario que posea la autorización adecuada para llevar a cabo las acciones siguientes:

- Llamar a la API de OS/400 que se indica en el ejemplo.
- Acceder a la información que se solicita.

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplo: ejemplo simple de recuperación de datos

Este ejemplo simple consta de dos partes:

- Código fuente PCML para llamar a QSYRUSRI
- Código fuente de programa Java para llamar a QSYRUSRI

Código fuente PCML para llamar a QSYRUSRI

```
<pcml version="1.0">

<!-- Fuente PCML para llamar a la API "Recuperar información de usuario"
(QSYRUSRI) -->

  <!-- Formato USRI0150 - Hay otros formatos disponibles -->
  <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>

  <!-- Programa QSYRUSRI y su lista de parámetros para recuperar el formato
USRI0100 -->
  <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>
```

Código fuente de programa Java para llamar a 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;

// Programa de ejemplo para llamar a la API "Recuperar información de
usuario" (QSYRUSRI)
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;        // Código de retorno de
ProgramCallDocument.callProgram()
    String msgId, msgText;     // Mensajes devueltos desde el servidor
    Object value;              // Valor de retorno de
ProgramCallDocument.getValue()

    System.setErr(System.out);

      // Construya AS400 sin parámetros, se solicitarán al usuario.
    as400System = new AS400();

                            try
        {
      // Quite la marca de comentario de la línea siguiente para obtener
información de depuración.
      //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

            System.out.println("Está empezando el ejemplo PCML.");
            System.out.println("    Se está construyendo ProgramCallDocument
para la API QSYRUSRI...");

      // Construya ProgramCallDocument.
      // El primer parámetro es el sistema al que se va a conectar.
      // El segundo parámetro es el nombre del recurso PCML. En este
ejemplo,
      // el archivo PCML serializado "qsyrusri.pcml.ser" o
      // el archivo fuente PCML "qsyrusri.pcml" debe estar en la vía de
acceso de clases.
            pcml = new ProgramCallDocument(as400System, "qsyrusri");
```

```
      // Establezca los parámetros de entrada. Varios parámetros tienen
valores por omisión
      // especificados en el fuente PCML. No es necesario establecerlos
mediante código Java.
            System.out.println("    Se están estableciendo parámetros de
entrada...");
            pcml.setValue("qsyrusri.receiverLength", new
Integer((pcml.getOutputsize("qsyrusri.receiver"))));

      // Petición para llamar a la API.
      // Se solicitará al usuario que inicie la sesión en el sistema.
            System.out.println("    Se está llamando a la API QSYRUSRI para
pedir información del usuario de inicio de sesión.");
            rc = pcml.callProgram("qsyrusri");

      // Si el código de retorno es false, se han recibido mensajes del
servidor.
      if (rc == false)
            {
      // Recupere la lista de mensajes del servidor.
               AS400Message[] msgs = pcml.getMessageList("qsyrusri");

              // Itere a través de los mensajes y escríbalos en la salida
estándar.
              for (int m = 0; m < msgs.length; m++)
                {
                  msgId = msgs[m].getID();
                  msgText = msgs[m].getText();
                    System.out.println("    " + msgId + " - " + msgText);
                }
                System.out.println("** Ha fallado la llamada a QSYRUSRI. Vea
los mensajes anteriores **");
       System.exit(0);
            }
      // Si el código de retorno ha sido true, la llamada a QSYRUSRI ha sido
satisfactoria
      // Escriba parte de los resultados en la salida estándar.
          else
            {
                value = pcml.getValue("qsyrusri.receiver.bytesReturned");
                System.out.println("        Bytes devueltos:      " +
value);
                value = pcml.getValue("qsyrusri.receiver.bytesAvailable");
                System.out.println("        Bytes disponibles:     " +
value);
                value = pcml.getValue("qsyrusri.receiver.userProfile");
                System.out.println("        Nombre de perfil:        " +
value);
                value =
pcml.getValue("qsyrusri.receiver.previousSignonDate");
                System.out.println("        Fecha de inicio de sesión
anterior:" + value);
                value =
pcml.getValue("qsyrusri.receiver.previousSignonTime");
                System.out.println("        Hora de inicio de sesión
anterior:" + value);
            }
         }
    catch(PcmlException e)
```

```
        {
      System.out.println(e.getLocalizedMessage());
      e.printStackTrace();
            System.out.println("*** Ha fallado la llamada a QSYRUSRI. ***");
       System.exit(0);
        }

       System.exit(0);
    } // Fin de main()

}
```

Ejemplo: recuperar una lista de información

Este ejemplo consta de dos partes:

- Código fuente PCML para llamar a QGYOLAUS
- Código fuente de programa Java para llamar a QGYOLAUS

Fuente PCML para llamar a QGYOLAUS

```
<pcml version="1.0">

<!-- Fuente PCML para llamar a la API "Abrir lista de usuarios autorizados"
(QGYOLAUS) -->

  <!-- Formato AUTU0150 - Hay otros formatos disponibles -->
  <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>

  <!-- Estructura para listar información (es común a todas las API de tipo
"Abrir lista") -->
  <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>

  <!-- El programa QGYOLAUS y su lista de parámetros para recuperar el
formato AUTU0150 -->
  <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>

  <!-- El programa QGYGTLE ha devuelto "registros" adicionales de la lista
creada por 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>

  <!-- El programa QGYCLST cierra la lista, liberando recursos en el
servidor -->
  <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>
```

Fuente de programa Java para llamar a 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;

// Programa de ejemplo para llamar a la API "Recuperar lista de usuarios
autorizados" (QGYOLAUS)
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;        // Código de retorno de
ProgramCallDocument.callProgram()
    String msgId, msgText;     // Mensajes devueltos desde el servidor
    Object value;              // Valor de retorno de
```

```
ProgramCallDocument.getValue()

    int[] indices = new int[1]; // Índices para acceder al valor de la
matriz
    int nbrRcds,                // Número de registros devueltos desde
QGYOLAUS y QGYGTLE
        nbrUsers;               // Número total de usuarios recuperados
    String listStatus;          // Estado de la lista en el servidor
    byte[] requestHandle = new byte[4];

    System.setErr(System.out);

    // Construya AS400 sin parámetros, se solicitarán al usuario
    as400System = new AS400();

                            try
    {
      // Quite la marca de comentario de la línea siguiente para obtener
información de depuración
      //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Está empezando el ejemplo PCML.");
      System.out.println("    Se está construyendo ProgramCallDocument para
la API QGYOLAUS...");

      // Construya ProgramCallDocument.
      // El primer parámetro es el sistema al que se va a conectar.
      // El segundo parámetro es el nombre del recurso PCML. En este
ejemplo,
      // el archivo PCML serializado "qgyolaus.pcml.ser" o
      // el archivo fuente PCML "qgyolaus.pcml" debe estar en la vía de
acceso de clases.
      pcml = new ProgramCallDocument(as400System, "qgyolaus");

      // Todos los parámetros de entrada tienen valores por omisión
especificados en el fuente PCML.
      // No es necesario establecerlos mediante código Java.

      // Petición para llamar a la API.
      // Se solicitará al usuario que inicie la sesión en el sistema.
      System.out.println("    Se está llamando a la API QGYOLAUS para
solicitar información del usuario de inicio de sesión.");
      rc = pcml.callProgram("qgyolaus");

      // Si el código de retorno es false, se han recibido mensajes del
servidor.
      if (rc == false)
      {
        // Recupere la lista de mensajes del servidor.
        AS400Message[] msgs = pcml.getMessageList("qgyolaus");

              // Itere a través de los mensajes y escríbalos en la salida
estándar.
              for (int m = 0; m < msgs.length; m++)
        {
                  msgId = msgs[m].getID();
                  msgText = msgs[m].getText();
             System.out.println("    " + msgId + " - " + msgText);
        }
        System.out.println("** Ha fallado la llamada a QGYOLAUS. Vea los
```

```
mensajes anteriores **");
       System.exit(0);
      }
      // Si el código de retorno ha sido true, la llamada a QGYOLAUS ha sido
satisfactoria
      // Escriba parte de los resultados en la salida estándar.
          else
      {
        boolean doneProcessingList = false;
        String programName = "qgyolaus";
        nbrUsers = 0;
        while (!doneProcessingList)
        {
          nbrRcds = pcml.getIntValue(programName +
".listInfo.rcdsReturned");
          requestHandle = (byte[]) pcml.getValue(programName +
".listInfo.rqsHandle");

          // Itere a través de la lista de usuarios
          for (indices[0] = 0; indices[0] < nbrRcds; indices[0]++)
          {
               value = pcml.getValue(programName + ".receiver.name",
indices);
               System.out.println("Usuario:  " + value);

               value = pcml.getValue(programName + ".receiver.description",
indices);
               System.out.println("\t\t" + value);
          }

          nbrUsers += nbrRcds;

          // Compruebe si se han recuperado todos los usuarios.
          // Si no se han recuperado todos, será necesario hacer nuevas
llamadas a "Obtener entradas de lista" (QGYGTLE)
          // para recuperar los demás usuarios de la lista.
          listStatus = (String) pcml.getValue(programName +
".listInfo.listStatus");
          if ( listStatus.equals("2")   // La lista está marcada como
"Completa"
            || listStatus.equals("3") ) // O la lista está marcada como
"Error al construir"
          {
            doneProcessingList = true;
          }
          else
          {
            programName = "qgygtle";

            // Establezca los parámetros de entrada para QGYGTLE
            pcml.setValue("qgygtle.requestHandle", requestHandle);
            pcml.setIntValue("qgygtle.startingRcd", nbrUsers + 1);

            // Llame a "Obtener entradas de lista" (QGYGTLE) para obtener
más usuarios de la lista.
            rc = pcml.callProgram("qgygtle");

      // Si el código de retorno es false, se han recibido mensajes del
servidor.
      if (rc == false)
```

```
            {
              // Recupere la lista de mensajes del servidor.
              AS400Message[] msgs = pcml.getMessageList("qgygtle");

              // Itere a través de los mensajes y escríbalos en la salida
estándar.
              for (int m = 0; m < msgs.length; m++)
              {
                  msgId = msgs[m].getID();
                  msgText = msgs[m].getText();
                  System.out.println("    " + msgId + " - " + msgText);
              }
              System.out.println("** Ha fallado la llamada a QGYGTLE. Vea
los mensajes anteriores **");
       System.exit(0);
            }
            // Si el código de retorno ha sido true, la llamada a QGYGTLE ha
sido satisfactoria.

          }
        }
        System.out.println("Número de usuarios devueltos:  " + nbrUsers);

        // Llame a la API "Cerrar lista" (QGYCLST).
        pcml.setValue("qgyclst.requestHandle", requestHandle);
        rc = pcml.callProgram("qgyclst");
      }
    }
    catch(PcmlException e)
    {
      System.out.println(e.getLocalizedMessage());
      e.printStackTrace();
      System.out.println("*** Ha fallado la llamada a QGYOLAUS. ***");
       System.exit(0);
    }

       System.exit(0);
  }
}
```

Ejemplo: recuperar datos multidimensionales

Este ejemplo consta de dos partes:

- Código fuente PCML para llamar a QZNFRTVE
- Código fuente de programa Java para llamar a QZNFRTVE

Fuente PCML para llamar a 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>
```

Fuente de programa Java para llamar a 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;

// Programa de ejemplo para llamar a la API "Recuperar exportaciones NFS"
(QZNFRTVE)
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;        // Código de retorno de
ProgramCallDocument.callProgram()
    String msgId, msgText;     // Mensajes devueltos desde el servidor
    Object value;              // Valor de retorno de
ProgramCallDocument.getValue()

    System.setErr(System.out);

    // Construya AS400 sin parámetros, se solicitarán al usuario
    as400System = new AS400();

    int[] indices = new int[2]; // Índices para acceder al valor de la
matriz.
    int nbrExports;             // Número de exportaciones devueltas.
    int nbrOfReadWriteHostNames, nbrOfRWHostNames,
        nbrOfRootHostNames,      nbrOfAccessHostnames, nbrOfHostOpts;

                           try
    {
      // Quite la marca de comentario de la línea siguiente para obtener
información de depuración
      // com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Está empezando el ejemplo PCML.");
      System.out.println("    Se está construyendo ProgramCallDocument para
la API QZNFRTVE...");

      // Construya ProgramCallDocument.
      // El primer parámetro es el sistema al que se va a conectar.
      // El segundo parámetro es el nombre del recurso PCML. En este
ejemplo,
      // el archivo PCML serializado "qznfrtve.pcml.ser" o el archivo
      // fuente PCML "qznfrtve.pcml" debe estar en la vía de acceso de
clases.
      pcml = new ProgramCallDocument(as400System, "qznfrtve");

      // Establezca los parámetros de entrada. Varios parámetros tienen
valores por omisión
      // especificados en el fuente PCML. No es necesario establecerlos
mediante código Java.
      System.out.println("    Se están estableciendo parámetros de
entrada...");
      pcml.setValue("qznfrtve.receiverLength", new Integer( (
```

```
pcml.getOutputsize("qznfrtve.receiver"))));

      // Petición para llamar a la API.
      // Se solicitará al usuario que inicie la sesión en el sistema.
      System.out.println("    Se está llamando a la API QZNFRTVE que
solicita exportaciones NFS.");
      rc = pcml.callProgram("qznfrtve");

      if (rc == false)
      {
        // Recupere la lista de mensajes del servidor.
        AS400Message[] msgs = pcml.getMessageList("qznfrtve");

        // Itere a través de los mensajes y escríbalos en la salida
estándar.
              for (int m = 0; m < msgs.length; m++)
        {
                  msgId = msgs[m].getID();
                  msgText = msgs[m].getText();
            System.out.println("    " + msgId + " - " + msgText);
        }
        System.out.println("** Ha fallado la llamada a QZNFRTVE. Vea los
mensajes anteriores **");
       System.exit(0);
      }
      // El código de retorno ha sido true, la llamada a QZNFRTVE ha sido
satisfactoria.
      // Escriba parte de los resultados en la salida estándar.
          else
      {
        nbrExports =
pcml.getIntValue("qznfrtve.returnedRcdsFdbkInfo.nbrOfNFSExportEntries");
        // Itere a través de la lista de exportaciones.
        for (indices[0] = 0; indices[0] < nbrExports; indices[0]++)
        {
          value = pcml.getValue("qznfrtve.receiver.pathName", indices);
          System.out.println("Nombre de vía = " + value);

          // Itere y escriba Nombres de sistema principal de
lectura/escritura para esta exportación.
          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("    Nombre de sistema principal de acceso de
lectura/escritura = " + value);
          }

          // Itere y escriba Nombres de sistema principal root para esta
exportación.
          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("    Nombre de sistema principal de acceso
```

```
root = " + value);
          }

          // Itere y escriba Nombres de sistema principal de acceso para
esta exportación.
          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("    Nombre de sistema principal de acceso =
" + value);
          }

          // Itere y escriba Opciones de sistema principal para esta
exportación.
          nbrOfHostOpts =
pcml.getIntValue("qznfrtve.receiver.nbrOfHostOptions", indices);
          for(indices[1] = 0; indices[1] < nbrOfHostOpts; indices[1]++)
          {
            System.out.println("    Opciones de sistema principal:");
            value =
pcml.getValue("qznfrtve.receiver.hostOptions.dataFileCodepage", indices);
            System.out.println("        Página de códigos de archivo de
datos = " + value);
            value =
pcml.getValue("qznfrtve.receiver.hostOptions.pathNameCodepage", indices);
            System.out.println("        Página de códigos de nombre de vía
de acceso = " + value);
            value =
pcml.getValue("qznfrtve.receiver.hostOptions.writeModeFlag", indices);
            System.out.println("        Distintivo de modalidad de escritura
= " + value);
            value = pcml.getValue("qznfrtve.receiver.hostOptions.hostName",
indices);
            System.out.println("        Nombre de sistema principal = " +
value);
          }
        } // Fin de iteración en bucle por la lista de exportaciones.
      } // El fin de llamada a QZNFRTVE ha sido satisfactorio.
    }
    catch(PcmlException e)
    {
      System.out.println(e.getLocalizedMessage());
      e.printStackTrace();
      System.exit(-1);
    }

       System.exit(0);
  } // Fin de main().
}
```

Ejemplos: clases ReportWriter

En esta sección figura una lista de los ejemplos de código que se proporcionan en toda la documentación de las clases ReportWriter.

JSPReportProcessor y PDFContext

- Ejemplo: cómo se utiliza JSPReportProcessor con PDFContext
- »Ejemplo: archivo JSP de ejemplo de JSPReportProcessor«

XSLReportProcessor y PCLContext

- Ejemplo: cómo se utiliza XSLReportProcessor con PCLContext
- »Ejemplo: archivo XML de ejemplo de XSLReportProcessor«
- »Ejemplo: archivo XSL de ejemplo de XSLReportProcessor«

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplo: cómo se utiliza JSPReportProcessor con PDFContext

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// El ejemplo siguiente (JSPRunReport) utiliza las clases JSPReportProcessor
y
// PDFContext para obtener datos de un URL especificado y convertir los
datos
// en el formato PDF. Después los datos se convierten en un documento PDF.
//
// Para ver el contenido de un archivo fuente JSP de ejemplo que puede
emplear con
// JSPRunReport, consulte JSPcust_table.jsp. También puede bajar un archivo
zip
// que contiene el archivo de ejemplo JSP. El archivo zip también contiene
los
// archivos de ejemplo XML y XSL que puede utilizar con el ejemplo de
XSLReportProcessor
// (PCLRunReport).
//
// Sintaxis de mandato:
//     java JSPRunReport <Url_jsp> <nombre_archivo_salida>
//
////////////////////////////////////////////////////////////////////////////

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;
```

```
        /** Especifique el URL que contiene los datos
            que desea utilizar en el informe. **/
        String JSPurl = args[0];
        URL jspurl = null;
  try {
              jspurl = new URL(JSPurl);
        }
        catch (MalformedURLException e)
        {}

        /** Obtenga el nombre de archivo PDF de salida. **/
        String filename = args[1];
  try {
        fileout = new FileOutputStream(filename);
        }
        catch (FileNotFoundException e)
        {}

        /** Configure el formato de página. **/
        Paper paper = new Paper();
        paper.setSize(612,792);
        paper.setImageableArea(18, 18, 576, 756);
        PageFormat pf = new PageFormat();
        pf.setPaper(paper);

        /** Cree un objeto PDFContext y convierta FileOutputStream
            como OutputStream. **/
        PDFContext pdfcontext = new PDFContext((OutputStream)fileout, pf);

        System.out.println( Preparado para analizar documento XSL );

        /** Cree el objeto JSPReportProcessor y establezca la plantilla
            en el JSP especificado. **/
        JSPReportProcessor jspprocessor = new
JSPReportProcessor(pdfcontext);
  try {
        jspprocessor.setTemplate(jspurl);
        }

        catch (NullPointerException np){
            String mes = np.getMessage();
            System.out.println(mes);
       System.exit(0);
            }

        /** Procese el informe. **/
  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);
```

```
    }
}
```

Ejemplo: archivo JSP de ejemplo de JSPReportProcessor

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
<?xml version="1.0"?>

<!--
  Copyright (c) 1999 The Apache Software Foundation. Reservados
  todos los derechos.
-->

<%@ 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 incluye el nombre del cliente, la dirección del
cliente, la ciudad
          // del cliente, la región administrativa del cliente y el código
postal del cliente.



          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;
   }
%>


<!-- Primera prueba de análisis y composición. -->
<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>

           <%
             // Añada fila a tabla.
             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>

        <%
        } // fin de fila
        %>

        </fo:table-body>
      </fo:table>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Ejemplo: cómo se utiliza XSLReportProcessor con PCLContext

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// El ejemplo siguiente (PCLRunReport) utiliza las clases
XSLPReportProcessor y
// PCLContext para obtener datos XML y convertir los datos al formato PCL.
// Después los datos se pasan como una corriente a una OutputQueue de
impresora.
//
// Para ver el contenido de los archivos fuente XML y XSL de ejemplo que
puede emplear
// con PCLRunReport, consulte realestate.xml y realestate.xsl. También puede
// bajar un archivo zip que contiene los archivos de ejemplo XML y XSL. El
archivo zip
// también contiene un archivo de ejemplo JSP que puede emplear con el
ejemplo
// JSPReportProcessor (JSPRunReport).
//
// Sintaxis de mandato:
//     java PCLRunReport <archivo_xml> <archivo_xsl>
//
////////////////////////////////////////////////////////////////////////////

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>";      /* Inserte el nombre de sistema
ISeries              */
        String user = "<user>";       /* Inserte el nombre de perfil de
usuario de ISeries */
        String pass = "<password>";   /* Inserte la contraseña de ISeries
*/

        AS400 system = new AS400(sys, user, pass);

        /* Inserte la cola de salida de ISeries */
        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)
        {}

        /** Configure el formato de página. **/
        Paper paper = new Paper();
        paper.setSize(612,792);
        paper.setImageableArea(18, 36, 576, 720);
        PageFormat pf = new PageFormat();
        pf.setPaper(paper);

        /** Cree un objeto PCLContext y convierta FileOutputStream
            como OutputStream **/
        PCLContext pclcontext = new PCLContext((OutputStream)fileout, pf);

        System.out.println("Preparado para analizar documento XSL");

        /** Cree el objeto XSLReportProcessor **/
        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);
            }
        /** Establezca la plantilla en el origen de datos XML especificado
**/
  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);
            }

         /** Procese el informe **/
   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);
     }

}
```

»

Ejemplo: archivo XML de ejemplo de XSLReportProcessor

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
<?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>
```

Ejemplo: archivo XSL de ejemplo de XSLReportProcessor

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
<?xml version="1.0"?>

<!-- Ejemplo de estilo de un documento inmobiliario imaginado. -->
<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>
```

Ejemplos: clases de recursos

En esta sección figura una lista de los ejemplos de código que se proporcionan en toda la documentación de las clases de recursos.

Resource y ChangeableResource

- Ejemplo: recuperar un valor de atributo de RUser, una subclase concreta de Resource
- Ejemplo: establecer valores de atributo para RJob, una subclase concreta de ChangeableResource
- Ejemplo: cómo se utiliza el código genérico para acceder a los recursos

ResourceList

- Ejemplo: obtener e imprimir el contenido de un objeto ResourceList
- Ejemplo: cómo se utiliza el código genérico para acceder a un objeto ResourceList
- Ejemplo: visualizar una lista de recursos en un servlet

Presentación

- Ejemplo: cómo se utilizan las presentaciones

Ejemplo: recuperar un valor de atributo de un objeto Resource

Una subclase concreta de Resource es com.ibm.as400.resource.RUser, que representa un usuario de iSeries. RUser da soporte a muchos ID de atributo, cada uno de los cuales se puede utilizar para obtener valores de atributo.

Este ejemplo recupera un valor de atributo de un objeto RUser:

```
   // Cree un objeto RUser para hacer referencia a un usuario específico.
   AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
   RUser user = new RUser(system, "AUSERID");

   // Obtenga el valor de atributo de descripción de texto.
   String textDescription =
(String)user.getAttributeValue(RUser.TEXT_DESCRIPTION);
```

Ejemplo: establecer valores de atributo para un objeto ChangeableResource

Una subclase concreta de ChangeableResource es com.ibm.as400.resource.RJob, que representa un trabajo del iSeries. RJob da soporte a muchos ID de atributo, cada uno de los cuales se puede utilizar para acceder a los valores de atributo. Este ejemplo establece dos valores de atributo para un objeto RJob:

```
// Cree un objeto RJob para hacer referencia a un trabajo específico.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJob job = new RJob(system, "AJOBNAME", "AUSERID", "AJOBNUMBER");

// Establezca el valor de atributo de formato de fecha.
job.setAttributeValue(RJob.DATE_FORMAT, RJob.DATE_FORMAT_JULIAN);

// Establezca el valor del atributo de ID de país o región.
job.setAttributeValue(RJob.COUNTRY_ID, RJob.USER_PROFILE);

// Comprometa ambos cambios de atributo.
job.commitAttributeChanges();
```

Ejemplo: cómo se utiliza el código genérico para acceder a los recursos

Puede escribir código genérico para trabajar con cualquier subclase de Resource o ChangeableResource. Este código puede mejorar la capacidad de reutilización y de mantenimiento y funcionará con las futuras subclases de Resource o ChangeableResource sin necesidad de modificación.

Cada uno de los atributos tiene asociado un objeto de metadatos de atributo (com.ibm.as400.resource.ResourceMetaData) que describe diversas propiedades del atributo. Estas propiedades incluyen si el atributo es o no de sólo lectura y cuáles son los valores posibles y por omisión.

A continuación figura un ejemplo de código genérico que imprime el valor de cada uno de los atributos soportados por un recurso:

```
void printAllAttributeValues(Resource resource) throws ResourceException
{
    // Obtenga los metadatos de atributo.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Itere en bucle por todos los atributos e imprima los valores.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
            Object attributeID = attributeMetaData[i].getID();
        Object value = resource.getAttributeValue(attributeID);
        System.out.println("Atributo " + attributeID + " = " + value);
    }
}
```

A continuación figura un ejemplo de código genérico que restablece todos los atributos de un objeto ChangeableResource a sus valores por omisión:

```
void resetAttributeValues(ChangeableResource resource) throws
ResourceException
{
    // Obtenga los metadatos de atributo.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Itere en bucle por todos los atributos.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        // Si el atributo es modificable (no es de sólo lectura),
        // restablezca su valor al valor por omisión.
        if (! attributeMetaData[i].isReadOnly())
        {
            Object attributeID = attributeMetaData[i].getID();
            Object defaultValue = attributeMetaData[i].getDefaultValue();
            resource.setAttributeValue(attributeID, defaultValue);
        }
    }

    // Comprometa todos los cambios de atributo.
    resource.commitAttributeChanges();
}
```

Ejemplos: listas de recursos

Los ejemplos siguientes muestran diversas formas de trabajar con listas de recursos:

- Ejemplo: obtener e imprimir el contenido de un objeto ResourceList
- Ejemplo: cómo se utiliza el código genérico para acceder a un objeto ResourceList
- Ejemplo: visualizar una lista de recursos en un servlet

Ejemplo: obtener e imprimir el contenido de un objeto ResourceList

Un ejemplo de una subclase concreta de ResourceList es com.ibm.as400.resource.RJobList, que representa una lista de trabajos de iSeries. RJobList da soporte a muchos ID de selección e ID de ordenación, cada uno de los cuales puede utilizarse para filtrar u ordenar la lista. Este ejemplo imprime el contenido de un objeto RJobList:

```
   // Cree un objeto RJobList para representar una lista de trabajos.
   AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
   RJobList jobList = new RJobList(system);

   // Filtre la lista para incluir únicamente los trabajos interactivos.
   jobList.setSelectionValue(RJobList.JOB_TYPE, RJob.JOB_TYPE_INTERACTIVE);

   // Ordene la lista por nombre de usuario y a continuación por nombre de
trabajo.
   Object[] sortValue = new Object[] { RJob.USER_NAME, RJob.JOB_NAME };
   jobList.setSortValue(sortValue);

   // Abra la lista y espere a que se complete.
   jobList.open();
   jobList.waitForComplete();

   // Lea e imprima el contenido de la lista.
   long length = jobList.getListLength();
   for(long i = 0; i < length; ++i)
   {
       System.out.println(jobList.resourceAt(i));
   }

   // Cierre la lista.
   jobList.close();
```

Ejemplo: cómo se utiliza el código genérico para trabajar con recursos

Además de utilizar subclases concretas directamente, puede escribir código genérico para trabajar con cualquier subclase de ResourceList. Este código puede mejorar la capacidad de reutilización y de mantenimiento y funcionará con las futuras subclases de ResourceList sin necesidad de modificación.

Ejemplo: imprimir el contenido de un objeto ResourceList

A continuación figura un ejemplo de código genérico que imprime parte del contenido de un objeto ResourceList:

```
   void printContents(ResourceList resourceList, long numberOfItems) throws
ResourceException
   {
       // Abra la lista y espere a que esté disponible el número
       // de elementos solicitados.
```

```
        resourceList.open();
        resourceList.waitForResource(numberOfItems);

        for(long i = 0; i < numberOfItems; ++i)
        {
            System.out.println(resourceList.resourceAt(i));
        }
    }
```

Ejemplo: cómo se utiliza ResourceMetaData para acceder a todos los atributos soportados por un recurso

Cada uno de los atributos tiene asociado un objeto de metadatos de atributo (com.ibm.as400.resource.ResourceMetaData) que describe diversas propiedades del atributo. Estas propiedades incluyen si el atributo es o no de sólo lectura y cuáles son los valores posibles y por omisión.

A continuación figura un ejemplo de código genérico que imprime el valor de cada uno de los atributos soportados por un recurso:

```
void printAllAttributeValues(Resource resource) throws ResourceException
{
    // Obtenga los metadatos de atributo.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Itere en bucle por todos los atributos e imprima los valores.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
            Object attributeID = attributeMetaData[i].getID();
        Object value = resource.getAttributeValue(attributeID);
        System.out.println("Atributo " + attributeID + " = " + value);
    }
}
```

Ejemplo: cómo se utiliza ResourceMetaData para restablecer cada uno de los atributos de un objeto ChangeableResource

A continuación figura un ejemplo de código genérico que restablece todos los atributos de un objeto ChangeableResource a sus valores por omisión:

```
void resetAttributeValues(ChangeableResource resource) throws
ResourceException
{
    // Obtenga los metadatos de atributo.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();


    // Itere en bucle por todos los atributos.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        // Si el atributo es modificable (no es de sólo lectura),
        // restablezca su valor al valor por omisión.
        if (! attributeMetaData[i].isReadOnly())
        {
            Object attributeID = attributeMetaData[i].getID();
            Object defaultValue = attributeMetaData[i].getDefaultValue();
            resource.setAttributeValue(attributeID, defaultValue);
        }
    }

    // Comprometa todos los cambios de atributo.
```

```
    resource.commitAttributeChanges();
}
```

Ejemplo: visualizar una lista de recursos en un servlet

Utilice la clase ResourceListRowData junto con la clase HTMLFormConverter o HTMLTableConverter para visualizar una lista de recursos en un servlet.

- HTMLFormConverter visualiza el contenido de la lista de recursos como una serie de formularios en la que cada formulario contiene valores de atributo para un recurso de la lista de recursos.
- HTMLTableConverter visualiza el contenido de la lista de recursos como una tabla en la que cada fila contiene información sobre un recurso de la lista de recursos.

Las columnas de un objeto ResourceListRowData se especifican como una matriz de ID de atributo de columna, mientras que cada fila representa un objeto de recurso.

```
   // Cree la lista de recursos. Este ejemplo crea una lista
   // de todos los mensajes de la cola de mensajes del
   // usuario actual.
   AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
   RMessageQueue messageQueue = new RMessageQueue(system,
RMessageQueue.CURRENT);

   // Cree el objeto ResourceListRowData. En este ejemplo
   // hay cuatro columnas en la tabla. La primera columna
   // contiene los iconos y los nombres de cada mensaje de la
   // cola de mensajes. Las demás columnas contienen el texto,
   // la gravedad y el tipo de cada mensaje.
   ResourceListRowData rowdata = new ResourceListRowData(messageQueue,
       new Object[] { null, RQueuedMessage.MESSAGE_TEXT,
RQueuedMessage.MESSAGE_SEVERITY,
                      RQueuedMessage.MESSAGE_TYPE } );

   // Cree los objetos HTMLTable y HTMLTableConverter que se
   // usarán para generar y personalizar las tablas HTML.
         HTMLTable table = new HTMLTable();
   table.setCellSpacing(6);
   table.setBorderWidth(8);

   HTMLTableConverter converter = new HTMLTableConverter();
   converter.setTable(table);
   converter.setUseMetaData(true);

   // Genere la tabla HTML.
   String[] html = converter.convert(rowdata);
         System.out.println(html[0]);
```

Ejemplos: RFML

En esta sección figura una lista de los ejemplos de código que se proporcionan en toda la documentación de RFML:

- Ejemplo: cómo se utiliza RFML en comparación con el uso de las clases Record de Toolbox para Java
- Ejemplo: archivo fuente RFML

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplo: cómo se utiliza una credencial de símbolo de perfil para intercambiar la identidad de la hebra de OS/400

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

El siguiente ejemplo de código muestra cómo se utiliza una credencial de símbolo de perfil para intercambiar la identidad de la hebra de OS/400 y trabajar en nombre de un determinado usuario:

```
     // Prepárese para trabajar con el sistema AS/400 local.
     AS400 system = new AS400("localhost", "*CURRENT", "*CURRENT");

     // Cree una ProfileTokenCredential de un solo uso con un tiempo de
espera de 60 segundos.
     // Es preciso sustituir un ID de usuario y una contraseña válidos.
     ProfileTokenCredential pt = new ProfileTokenCredential();
     pt.setSystem(system);
     pt.setTimeoutInterval(60);
     pt.setTokenType(ProfileTokenCredential.TYPE_SINGLE_USE);

     pt.setToken("USERID", "PASSWORD");

     // Intercambie la identidad de hebra de OS/400 y recupere una
credencial para
     // más adelante hacer un nuevo intercambio de regreso a la identidad
original.
     AS400Credential cr = pt.swap(true);

     // Trabaje bajo la identidad intercambiada en este momento.

     // Intercambie de nuevo para regresar a la identidad de hebra original
de OS/400.
     cr.swap();

     // Borre las credenciales.
     cr.destroy();
     pt.destroy();
```

Ejemplos de las clases de servlets

Los ejemplos que figuran a continuación muestran algunas maneras de cómo se pueden utilizar las clases de servlets:

- Ejemplo: cómo se utiliza la clase ListRowData
- Ejemplo: cómo se utiliza la clase RecordListRowData
- Ejemplo: cómo se utiliza la clase SQLResultSetRowData
- Ejemplo: cómo se utiliza la clase HTMLFormConverter
- Ejemplo: cómo se utiliza la clase ListMetaData
- Ejemplo: cómo se utiliza la clase SQLResultSetMetaData
- Ejemplo: visualizar una lista de recursos en un servlet

También se pueden utilizar juntas las clases de servlets y las clases HTML, como en este ejemplo.

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplo: cómo se utiliza ListRowData

Este ejemplo consta de tres partes:

- Fuente Java que muestra cómo funciona la clase ListRowData
- Fuente HTML generado a partir del fuente Java mediante HTMLTableConverter
- Visualización en un navegador del HTML generado

Fuente Java que muestra cómo funciona la clase ListRowData

```
         // Acceda a una cola de datos existente que no esté vacía.
         KeyedDataQueue dq = new KeyedDataQueue(systemObject_,
"/QSYS.LIB/MYLIB.LIB/MYDQ.DTAQ");

         // Cree un objeto de metadatos.
         ListMetaData metaData = new ListMetaData(2);

         // Establezca que la primera columna sea el ID de cliente.
         metaData.setColumnName(0, "ID de cliente");
         metaData.setColumnLabel(0, "ID de cliente");
         metaData.setColumnType(0, RowMetaDataType.STRING_DATA_TYPE);

         // Establezca que la segunda columna sea el pedido que se ha de
procesar.
         metaData.setColumnName(1, "Número de pedido");
         metaData.setColumnLabel(1, "Número de pedido");
         metaData.setColumnType(1, RowMetaDataType.STRING_DATA_TYPE);

         // Cree un objeto ListRowData.
         ListRowData rowData = new ListRowData();
         rowData.setMetaData(metaData);

         // Obtenga las entradas a partir de la cola de datos.
         KeyedDataQueueEntry data = dq.read(key, 0, "EQ");
         while (data != null)
         {
            // Añada la entrada de la cola al objeto de datos de fila.
            Object[] row = new Object[2];
            row[0] = new String(key);
            row[1] = new String(data.getData());
            rowData.addRow(row);

            // Obtenga otra entrada de la cola.
            data = dq.read(key, 0, "EQ");
         }

         // Cree un objeto conversor HTML y convierta los datos de fila
(rowData) a HTML.
         HTMLTableConverter conv = new HTMLTableConverter();
         conv.setUseMetaData(true);
         HTMLTable[] html = conv.convertToTables(rowData);

         // Visualice la salida del conversor.
         System.out.println(html[0]);
```

Fuente HTML generado a partir del fuente Java mediante HTMLTableConverter

Al utilizar la clase HTMLTableConverter en el ejemplo de fuente Java anterior se genera el siguiente código HTML.

```
<table>
<tr>
<th>ID de cliente</th>
<th>Número de pedido</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>
```

Visualización en un navegador del HTML generado

La tabla que hay a continuación muestra cómo se visualiza el código fuente HTML en un navegador.

ID de cliente	Número de pedido
777-53-4444	12345-XYZ
777-53-4444	56789-ABC

Ejemplo: cómo se utiliza RecordListRowData

Este ejemplo consta de tres partes:

- Fuente Java que muestra cómo funciona la clase RecordListRowData
- Fuente HTML generado a partir del fuente Java mediante HTMLTableConverter
- Visualización en un navegador del HTML generado

Fuente Java que muestra cómo funciona la clase RecordListRowData

```
         // Cree un objeto de servidor.
         AS400 mySystem = new AS400 ("mySystem.myComp.com", "UserId",
"Password");

         // Obtenga el nombre de la vía de acceso del archivo.
         QSYSObjectPathName file = new QSYSObjectPathName(myLibrary, myFile,
"%first%", "mbr");
         String ifspath = file.getPath();

         // Cree un objeto archivo que represente el archivo.
         SequentialFile sf = new SequentialFile(mySystem, ifspath);

         // Recupere el formato de registro del archivo.
         AS400FileRecordDescription recordDescription = new
AS400FileRecordDescription(mySystem, ifspath);
         RecordFormat recordFormat =
recordDescription.retrieveRecordFormat()[0];

         // Establezca el formato de registro del archivo.
         sf.setRecordFormat(recordFormat);

         // Obtenga los registros del archivo.
         Record[] records = sf.readAll();

         // Cree un objeto RecordListRowData y añádale los registros.
         RecordListRowData rowData = new RecordListRowData(recordFormat);

         for (int i=0; i < records.length; i++)
         {
            rowData.addRow(records[i]);
         }

         // Cree un objeto conversor HTML y convierta los datos de fila
(rowData) a HTML.
         HTMLTableConverter conv = new HTMLTableConverter();
         conv.setMaximumTableSize(3);
         HTMLTable[] html = conv.convertToTables(rowData);

         // Visualice la primera tabla HTML generada por el conversor.
         System.out.println(html[0]);
```

Fuente HTML generado a partir del fuente Java mediante HTMLTableConverter

Al utilizar la clase HTMLTableConverter en el ejemplo de fuente Java anterior se genera el siguiente código HTML.

```
<table>
<tr>
<th>NÚMCLI</th>
<th>APELLIDO</th>
<th>INIC</th>
<th>CALLE</th>
<th>CIUDAD</th>
<th>ESTADO</th>
<th>CÓDPOST</th>
<th>LMTCDT</th>
<th>CÓDCARG</th>
<th>SALDO</th>
<th>CDTVENC</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>
```

Visualización en un navegador del HTML generado

La tabla que hay a continuación muestra cómo se visualiza el código fuente HTML en un navegador.

NÚMCLI	APELLIDO	INIC	CALLE	CIUDAD	ESTADO	CÓDPOST	LMTCDT	CÓDCARG	SALDO	CDTVENC

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

Ejemplo: cómo se utiliza SQLResultSetRowData

Este ejemplo consta de tres partes:

- Fuente Java que muestra cómo funciona la clase SQLResultSetRowData
- Fuente HTML generado a partir del fuente Java mediante HTMLTableConverter
- Visualización en un navegador del HTML generado

Fuente Java que muestra cómo funciona la clase SQLResultSetRowData

```
         // Cree un objeto de servidor.
         AS400 mySystem = new AS400 ("mySystem.myComp.com", "UserId",
"Password");

         // Registre y obtenga una conexión con la base de datos.
         DriverManager.registerDriver(new
com.ibm.as400.access.AS400JDBCDriver());
         Connection connection = DriverManager.getConnection("jdbc:as400://"
+ mySystem.getSystemName());

         // Ejecute una sentencia SQL y obtenga el conjunto de resultados.
         Statement statement = connection.createStatement();
         statement.execute("select * from qiws.qcustcdt");
         ResultSet resultSet = statement.getResultSet();

         // Cree el objeto SQLResultSetRowData e inicialice con el conjunto
de resultados.
         SQLResultSetRowData rowData = new SQLResultSetRowData(resultSet);

         // Cree un objeto tabla HTML para que lo utilice el conversor.
         HTMLTable table = new HTMLTable();

         // Establezca cabeceras de columna descriptivas.
         String[] headers = {"Número cliente", "Apellido", "Iniciales",
                                               "Calle", "Ciudad", "Estado",
"Código postal",
                                               "Límite crédito", "Código
cargo", "Saldo",
                                               "Crédito vencido"};
         table.setHeader(headers);

         // Establezca varias opciones de formato dentro de la tabla.
         table.setBorderWidth(2);
         table.setCellSpacing(1);
         table.setCellPadding(1);

         // Cree un objeto conversor HTML y convierta los datos de fila
(rowData) a HTML.
         HTMLTableConverter conv = new HTMLTableConverter();
         conv.setTable(table);
         HTMLTable[] html = conv.convertToTables(rowData);

         // Visualice la tabla HTML generada por el conversor.
         System.out.println(html[0]);
```

Fuente HTML generado a partir del fuente Java mediante HTMLTableConverter

Al utilizar la clase HTMLTableConverter en el ejemplo de fuente Java anterior se genera el siguiente código HTML.

```
<table border="2" cellpadding="1" cellspacing="1">
<tr>
<th>Número cliente</th>
<th>Apellido</th>
<th>Iniciales</th>
<th>Calle</th>
<th>Ciudad</th>
<th>Estado</th>
<th>Código postal</th>
<th>Límite crédito</th>
<th>Código cargo</th>
<th>Saldo</th>
<th>Crédito vencido</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>
```

Visualización en un navegador del HTML generado

La tabla que hay a continuación muestra cómo se visualiza el código fuente HTML en un navegador.

Número cliente	Apellido	Iniciales	Calle	Ciudad	Estado	Código postal	Límite crédito	Código cargo	Saldo	Crédito vencido
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

Ejemplo: cómo se utiliza HTMLFormConverter

Mientras ejecuta un servidor Web con soporte para servlets, compile y ejecute el ejemplo siguiente para ver cómo funciona la clase HTMLFormConverter:

```
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;

/**
 * Ejemplo de cómo se utiliza la clase HTMLFormConverter en un servlet.
 */
public class HTMLFormConverterExample extends HttpServlet

{
   private String userId_ = "myUserId";
   private String password_ = "myPwd";
    private AS400 system_;

   private Connection databaseConnection_;



   // Haga un borrado antes de volver al formulario HTML principal.
   public void cleanup()
   {
                            try
      {
         // Cierre la conexión con la base de datos.
            if (databaseConnection_ != null)
         {
        databaseConnection_.close();
            databaseConnection_ = null;
```

```
      }
   }
     catch (Exception e)
   {
   e.printStackTrace();
   }
}


        // Convierta los datos de fila a HTML formateado.
private HTMLTable[] convertRowData(SQLResultSetRowData rowData)
{
                        try
   {
      // Cree el conversor, el cual generará HTML a partir del
   // conjunto de resultados devueltos por la consulta de base de datos.
      HTMLFormConverter converter = new HTMLFormConverter();

      // Establezca los atributos del formulario.
      converter.setBorderWidth(3);
      converter.setCellPadding(2);
      converter.setCellSpacing(4);

      // Convierta los datos de fila a HTML.
      HTMLTable[] htmlTable = converter.convertToForms(rowData);
      return htmlTable;
   }
     catch (Exception e)
   {
   e.printStackTrace();
     return null;
   }

}


// Devuelva la respuesta al cliente.
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
 throws IOException, ServletException
{
   response.setContentType("text/html");
 PrintWriter out = response.getWriter();

 out.println(showHtmlMain());
      out.close();
}


// Maneje los datos enviados al formulario.
  public void doPost (HttpServletRequest request, HttpServletResponse
response)
       throws ServletException, IOException
{
   SQLResultSetRowData rowData = new SQLResultSetRowData();
   HTMLTable[] htmlTable = null;

   // Obtenga el objeto sesión actual o cree uno si es necesario.
 HttpSession session = request.getSession(true);
```

```
    ServletOutputStream out = response.getOutputStream();

      response.setContentType("text/html");

    Hashtable parameters = getRequestParameters (request);

      // Recupere los valores de los datos de fila y los valores de tabla
HTML de esta sesión.
      rowData = (SQLResultSetRowData) session.getValue("sessionRowData");
      htmlTable = (HTMLTable[]) session.getValue("sessionHtmlTable");

      // Si es la primera vez que se pasa, mostrar el primer registro.
      if (parameters.containsKey("getRecords"))
      {
         rowData = getAllRecords(parameters, out);

         if (rowData != null)
         {
            // Establezca el valor de los datos de fila para esta sesión.
            session.putValue("sessionRowData", rowData);

            // Sitúese en el primer registro.
         rowData.first();

           // Convierta los datos de fila a HTML formateado.
            htmlTable = convertRowData(rowData);

            if (htmlTable != null)
            {
         rowData.first();
               session.putValue("sessionHtmlTable", htmlTable);
         out.println(showHtmlForRecord(htmlTable, 0));
            }
         }
      }
      // Si se ha pulsado el botón "Volver a principal", volver al
formulario HTML principal.
      else if (parameters.containsKey("returnToMain"))
      {
         session.invalidate();
         cleanup();
    out.println(showHtmlMain());
      }
      // Si se ha pulsado el botón "Primero", mostrar el primer registro.
      else if (parameters.containsKey("getFirstRecord"))
      {
         rowData.first();
         out.println(showHtmlForRecord(htmlTable, 0));
      }
      // Si se ha pulsado el botón "Anterior", mostrar el registro anterior.
      else if (parameters.containsKey("getPreviousRecord"))
      {
         if (!rowData.previous())
         {
         rowData.first();
         }
         out.println(showHtmlForRecord(htmlTable,
rowData.getCurrentPosition()));
      }
      // Si se ha pulsado el botón "Siguiente", mostrar el registro
```

```
siguiente.
      else if (parameters.containsKey("getNextRecord"))
      {
         if (!rowData.next())
         {
         rowData.last();
         }
         out.println(showHtmlForRecord(htmlTable,
rowData.getCurrentPosition()));
      }
      // Si se ha pulsado el botón "Último", mostrar el último registro.
      else if (parameters.containsKey("getLastRecord"))
      {
         rowData.last();
         out.println(showHtmlForRecord(htmlTable,
rowData.getCurrentPosition()));
      }
      // Si no se ha producido ninguno de los casos anteriores, debe haber
un error.
          else
      {
         out.println(showHtmlForError("Se ha producido un error interno.
Parámetros inesperados."));
      }

      // Guarde el valor de los datos de fila de esta sesión para que se
actualice
      // la posición actual en el objeto asociado a esta sesión.
      session.putValue("sessionRowData", rowData);

      // Cierre la corriente de datos de salida.
         out.close();
   }


   // Obtenga todos los registros del archivo entrado por el usuario.
   private SQLResultSetRowData getAllRecords(Hashtable parameters,
ServletOutputStream out)
   throws IOException
   {
      SQLResultSetRowData records = null;

                           try
      {
         // Obtenga el nombre del sistema, de la biblioteca y del archivo a
partir de la lista de parámetros.
         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("No es válido el nombre del
sistema, del archivo o de la biblioteca."));
         }
          else
         {
            // Obtenga la conexión con el servidor.
            getDatabaseConnection (sys, out);
```

```
            if (databaseConnection_ != null)
            {
      Statement sqlStatement = databaseConnection_.createStatement();

               // Consulte la base de datos para obtener el conjunto de
resultados.
               String query = "SELECT * FROM " + lib + "." + file;
      ResultSet rs = sqlStatement.executeQuery (query);

               boolean rsHasRows = rs.next();  // sitúe el cursor en la
primera fila.

               // Mostrar un mensaje de error si el archivo no contiene
ningún registro; en caso
               // contrario, establecer los datos de fila en los datos del
conjunto de resultados.
               if (!rsHasRows)
               {
                  out.println(showHtmlForError("No hay registros en el
archivo."));
               }
          else
               {
                  records = new SQLResultSetRowData (rs);
               }

               // No cierre Statement antes de terminar de usar ResultSet
               // porque de lo contrario se producirían anomalías.
      sqlStatement.close();
            }
         }
      }
        catch (Exception e)
      {
      e.printStackTrace();
            out.println(showHtmlForError(e.toString()));
      }

      return records;
   }


   // Establezca una conexión de base de datos.
   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()));
         }
      }
```

```
 }


 // Obtiene los parámetros a partir de una petición de servlet HTTP.
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;
 }


 // Obtenga la información de servlet.
  public String getServletInfo()
 {
    return "HTMLFormConverterExample";
 }


 // Siga los pasos de inicialización.
  public void init(ServletConfig config)
 {
                          try
    {
     super.init(config);

       // Registre el controlador JDBC.
                          try
       {
          DriverManager.registerDriver(new AS400JDBCDriver());
       }
      catch (Exception e)
       {
          System.out.println("Controlador JDBC no encontrado");
       }
    }
      catch (Exception e)
    {
    e.printStackTrace();
    }
 }


 // Establezca la información de cabecera de página.
 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 ();
 }


 // Mostrar la página HTML con la debida información de error.
```

```
   private String showHtmlForError(String message)
   {
      String title = "Error";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));
                           try
      {
      // Cree el objeto formulario HTML.
         HTMLForm errorForm = new HTMLForm("HTMLFormConverterExample");

      // Configúrelo de tal manera que se llame a doPost() cuando se someta
el formulario.
         errorForm.setMethod(HTMLForm.METHOD_POST);

         // Cree un panel de una sola columna al que se añadirán los
elementos HTML.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

         // Cree el elemento de texto para el error y añádalo al panel.
         HTMLText text = new HTMLText(message);
    text.setBold(true);
         text.setColor(Color.red);
      grid.addElement(text);

         // Cree el botón para volver a principal y añádalo al panel.
         grid.addElement(new SubmitFormInput("returnToMain", "Volver a
principal"));

         // Añada el panel al formulario HTML.
         errorForm.addElement(grid);

         page.append(errorForm.toString());
      }
        catch (Exception e)
      {
      e.printStackTrace();
      }
      page.append("</body></html>");
    return page.toString();
   }


   // Mostrar el formulario HTML para un registro individual.
   private String showHtmlForRecord(HTMLTable[] htmlTable, int position)
   {
      String title = "Ejemplo de HTMLFormConverter";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));

      page.append("<h1>" + title + "</h1>");

                           try
      {
      // Cree el objeto formulario HTML.
         HTMLForm recForm = new HTMLForm("HTMLFormConverterExample");

      // Configúrelo de tal manera que se llame a doPost() cuando se someta
el formulario.
         recForm.setMethod(HTMLForm.METHOD_POST);
```

```
      // Defina un diseño de panel de una sola columna, en el que
      // deban disponerse los elementos HTML generados.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

         // Cree y añada un pie de tabla que haga un seguimiento del
registro actual.
         HTMLText recNumText = new HTMLText("Número de registro: " +
(position + 1));
         recNumText.setBold(true);
         grid.addElement(recNumText);

         // Defina un diseño de panel de dos columnas, en el que deban
disponerse
         // la tabla y el texto de comentario de la salida del conversor.
         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);

         // Añada la línea de tabla al panel.
         grid.addElement(tableGrid);

         // Defina un diseño de panel de una sola fila, en el que deban
disponerse
         // los botones que permiten moverse por la lista de registros.
         LineLayoutFormPanel buttonLine = new LineLayoutFormPanel();
         buttonLine.addElement(new SubmitFormInput("getFirstRecord",
"Primero"));
         buttonLine.addElement(new SubmitFormInput("getPreviousRecord",
"Anterior"));
         buttonLine.addElement(new SubmitFormInput("getNextRecord",
"Siguiente"));
         buttonLine.addElement(new SubmitFormInput("getLastRecord",
"Último"));

         // Defina otro diseño de panel de una sola fila para el botón
Volver a principal.
         LineLayoutFormPanel returnToMainLine = new LineLayoutFormPanel();
         returnToMainLine.addElement(new SubmitFormInput("returnToMain",
"Volver a principal"));

         // Añada al panel cuadriculado las líneas que contienen los
botones.
         grid.addElement(buttonLine);
         grid.addElement(returnToMainLine);

         // Añada el panel al formulario.
         recForm.addElement(grid);

         // Añada el formulario a la página HTML.
         page.append(recForm.toString());
      }
        catch (Exception e)
      {
      e.printStackTrace();
      }
```

```
      page.append("</body></html>");
     return page.toString();
   }


   // Mostrar el formulario HTML principal (solicite entrada para nombre de
sistema, archivo y biblioteca).
   private String showHtmlMain()
   {
      String title = "Ejemplo de HTMLFormConverter";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));

      page.append("<h1>" + title + "</h1>");


      // Cree el objeto formulario HTML.
      HTMLForm mainForm = new HTMLForm("HTMLFormConverterExample");

                            try
      {
      // Configúrelo de tal manera que se llame a doPost() cuando se someta
el formulario.
      mainForm.setMethod(HTMLForm.METHOD_POST);

         // Añada una descripción corta al formulario.
         HTMLText desc = new HTMLText("<P>Este ejemplo usa la clase
HTMLFormConverter " +
                                      "para convertir los datos recuperados
de un archivo de " +
                                      "servidor. El conversor genera una
matriz de tablas " +
                                      "HTML. Cada entrada de la matriz es un
registro del " +
                                      "archivo.  " +
                                      "Los registros se visualizan de uno en
uno y hay " +
                                      "botones que permiten moverse hacia
delante o hacia " +
                                      "atrás en la lista de
registros.</P>");
         mainForm.addElement(desc);

         // Añada instrucciones al formulario.
         HTMLText instr = new HTMLText("<P>Por favor, entre el nombre del
servidor, " +
                                       "y el nombre del archivo y de la
biblioteca " +
                                       "a los que desea acceder. Luego pulse
el botón " +
                                       "Mostrar registros para
continuar.</P>");
         mainForm.addElement(instr);


         // Cree un panel con un diseño cuadriculado y añada los campos de
entrada de sistema, archivo y biblioteca.
  GridLayoutFormPanel panel = new GridLayoutFormPanel(2);

         LabelFormElement sysPrompt = new LabelFormElement("Servidor: ");
```

```
         TextFormInput system = new TextFormInput("System");
         system.setSize(10);

         LabelFormElement filePrompt = new LabelFormElement("Nombre de
archivo: ");
         TextFormInput file = new TextFormInput("File");
         file.setSize(10);

         LabelFormElement libPrompt = new LabelFormElement("Nombre de
biblioteca: ");
         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);

         // Añada el panel al formulario.
         mainForm.addElement(panel);

         // Cree el botón de someter y añádalo al formulario.
         mainForm.addElement(new SubmitFormInput("getRecords", "Mostrar
registros"));
      }
        catch (Exception e)
      {
      e.printStackTrace();
      }

    page.append(mainForm.toString());
      page.append("</body></html>");

    return page.toString();
   }

}
```

El HTML generado por el ejemplo anterior ofrece este aspecto:

```
<table border="0">
<tr>
<td><b>Número de registro: 1</b></td>
</tr>
<tr>
<td><table border="0">
<tr>
<td><table border="3" cellpadding="2" cellspacing="4">
<tr>
<th>NÚMCLI</th>
<td>839283</td>
</tr>
<tr>
<th>APELLIDO</th>
<td>Jones   </td>
</tr>
<tr>
```

```
<th>INIC</th>
<td>B D</td>
</tr>
<tr>
<th>CALLE</th>
<td>21B NW 135 St</td>
</tr>
<tr>
<th>CIUDAD</th>
<td>Clay  </td>
</tr>
<tr>
<th>ESTADO</th>
<td>NY</td>
</tr>
<tr>
<th>CÓDPOST</th>
<td>13041</td>
</tr>
<tr>
<th>LMTCDT</th>
<td>400</td>
</tr>
<tr>
<th>CÓDCARG</th>
<td>1</td>
</tr>
<tr>
<th>SALDO</th>
<td>100,00</td>
</tr>
<tr>
<th>CDTVENC</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="Primero" />
<input type="submit" name="getPreviousRecord" value="Anterior" />
<input type="submit" name="getNextRecord" value="Siguiente" />
<input type="submit" name="getLastRecord" value="Último" />
<br />
</td>
</tr>
<tr>
<td><input type="submit" name="returnToMain" value="Volver a principal" />
<br />
</td>
</tr>
</table>
</form>
```

Ejemplo de "Luces encendidas" para las clases HTML y servlet

Este ejemplo muestra cómo funcionan las clases HTML y servlet. Es una visión general. Para ver este ejemplo, compílelo y ejecútelo con un servidor Web y un navegador en marcha.

```
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.*;

/*
Un ejemplo de cómo se utilizan las clases de Toolbox en un servlet.

Esquemas de las bases de datos SQL en el servidor:

Archivo  . . . .   LICENSES
Biblioteca . .     LIGHTSON

 Campo            Tipo          Longitud Nulos
 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


Archivo  . . . .   REPORTS
Biblioteca . .     LIGHTSON

 Campo            Tipo          Longitud Nulos
 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_;  // contraseña del servidor y de la base de
datos SQL.
  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 { // Ninguno de los anteriores, por lo que presupondremos que el
usuario ha
           // rellenado un formulario y está sometiendo información. Tome la
           // información entrante y realice la acción solicitada.

      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 (interno): " +
                             "Ni informar, ni registrar, " +
                             "ni desregistrar, ni listar registrados ni listar
reportados."));
      }
    }

    out.close(); // Cierre la corriente de datos de salida.

  }

  // Obtiene los parámetros a partir de una petición de servlet HTTP y los
  // empaqueta en una tabla hash por comodidad.
  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;
  }


  // Elimina blancos y guiones de un objeto String y lo pone todo en
mayúsculas.
  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();
  }

  // Compone una lista de series entre comillas simples.
  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 "Servlet Luces encendidas";
  }

  private AS400 getSystem ()
  {
                            try
    {
      if (system_ == null)
      {
        system_ = new AS400();

        // Nota: sería mejor obtener estos valores
        // de un archivo de propiedades.
        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);

         // Registre el controlador JDBC.
  try {
      java.sql.DriverManager.registerDriver (new
com.ibm.as400.access.AS400JDBCDriver ());
      }
        catch (Exception e)
      {
        System.out.println("Controlador JDBC no encontrado");
      }


    }
                    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: número de matrícula no
especificado.\n");

    if (eMailAddress == null || eMailAddress.length() == 0)
      acknowledgement.append ("Error: dirección de correo-e para
notificación no especificada.\n");

    if (acknowledgement.length() == 0)
    {
                           try
      {
        // Inserte el nuevo número de matrícula y la nueva dirección de
correo-e en la base de datos.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

        // Emita la petición.
        String cmd = "INSERT INTO LICENSES " +
          "(LICENSE, E_MAIL) VALUES (" +
          quoteList (new String[] {licenseNum, eMailAddress} ) +
          ")";
        sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

        // Acuse el recibo de la petición.
        acknowledgement.append ("El número de matrícula " + licenseNum + "
se ha registrado.");
        acknowledgement.append ("La dirección de correo-e para notificación
es: " + 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: número de matrícula no
especificado.\n");

    if (acknowledgement.length() == 0)
    {
                            try
      {
        // Elimine de la base de datos el número de matrícula y la dirección
de correo-e especificados.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

        // Suprima la(s) fila(s) de la base de datos LICENSES.
        String cmd = "DELETE FROM LICENSES WHERE LICENSE = '" + licenseNum +
"'";
        sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

        // Acuse el recibo de la petición.
        acknowledgement.append ("El número de matrícula " + licenseNum + "
se ha desregistrado.");
      }
      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: número de matrícula no
especificado.");

    if (acknowledgement.length() == 0)
    {
                            try
      {
        // Informe de "luces encendidas" para un vehículo especificado.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

        // Añada una entrada a la base de datos REPORTS.
        String cmd = "INSERT INTO REPORTS " +
          "(LICENSE, LOCATION, COLOR, CATEGORY) VALUES (" +
          quoteList (new String[] {licenseNum, location, color, category} )
+
          ")";
```

```
        sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

        // Acuse el recibo de la petición.
        acknowledgement.append ("El número de matrícula " + licenseNum +
                                " se ha anotado en el informe. ¡Gracias!");
      }
      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 = "Acuse de recibo";
    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", "Inicio"));
      form.addElement(grid);
      page.append(form.toString());
    }
      catch (Exception e) { e.printStackTrace (); }
    page.append("</body></html>");
    return page.toString();
  }


  private String showHtmlMain ()
  {
    String title = "Herramienta Luces encendidas";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));

    page.append("<h1>" + title + "</h1>");


      // Cree el objeto formulario HTML.
      HTMLForm mainForm = new HTMLForm("LightsOn");
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

  try {
      // Configúrelo de tal manera que se llame a doPost() cuando se someta
el formulario.
```

```
      mainForm.setMethod(HTMLForm.METHOD_POST);

      // Cree algunos botones.
      grid.addElement(new SubmitFormInput("askingToReport",
                                          "Informar de un vehículo con las
luces encendidas"));
      grid.addElement(new SubmitFormInput("askingToRegister",
                                          "Registrar mi número de
matrícula"));
      grid.addElement(new SubmitFormInput("askingToUnregister",
                                          "Desregistrar mi número de
matrícula"));
      grid.addElement(new SubmitFormInput("askingToListRegistered",
                                          "Listar todas las matrículas
registradas"));
      grid.addElement(new SubmitFormInput("askingToListReported",
                                          "Listar todos los vehículos con
las luces encendidas"));

        mainForm.addElement(grid);
      }
        catch (Exception e) { e.printStackTrace (); }

      page.append(mainForm.toString());
      page.append("</body></html>");

      return page.toString();
    }


    private String showHtmlForReporting ()
    {
      String title = "Informar de un vehículo con las luces encendidas";
      StringBuffer page = new StringBuffer();
      page.append (showHeader (title));

      page.append("<h1>" + title + "</h1>");

        // Cree el objeto formulario HTML.
      HTMLForm reportForm = new HTMLForm("LightsOn");
      GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

    try {
        // Configúrelo de tal manera que se llame a doPost() cuando se someta
el formulario.
        reportForm.setMethod(HTMLForm.METHOD_POST);

        TextFormInput licenseNum = new TextFormInput("licenseNum");
        licenseNum.setSize(10);
        licenseNum.setMaxLength(10);

        // Añada elementos al formulario de línea.
        grid.addElement(new LabelFormElement("Número de matrícula de
vehículo:"));
        grid.addElement(licenseNum);

        // Cree un grupo de botones de selección y añada los botones de
selección.
        RadioFormInputGroup colorGroup = new RadioFormInputGroup("color");
```

```
      colorGroup.add("color", "white", "blanco", true);
      colorGroup.add("color", "black", "negro", false);
      colorGroup.add("color", "gray", "gris", false);
      colorGroup.add("color", "red", "rojo", false);
      colorGroup.add("color", "yellow", "amarillo", false);
      colorGroup.add("color", "green", "verde", false);
      colorGroup.add("color", "blue", "azul", false);
      colorGroup.add("color", "brown", "marrón", false);

      // Cree una lista de selección para la categoría del vehículo.
      SelectFormElement category = new SelectFormElement("category");
      category.addOption("sedan", "sedán", true);
      category.addOption("convertible", "convertibl"); // campo de 10
caracteres en DB
      category.addOption("truck", "camión");
      category.addOption("van", "furgoneta");
      category.addOption("SUV", "SUV");
      category.addOption("motorcycle", "moto");
      category.addOption("other", "otras");

      // Cree una lista de selección para la ubicación del vehículo (número
del edificio).
      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("Tipo de vehículo:"));
      grid.addElement(category);

      grid.addElement(new LabelFormElement("Edificio:"));
      grid.addElement(location);

      grid.addElement(new SubmitFormInput("submittingReport", "Someter
informe"));
      grid.addElement(new SubmitFormInput("returningToMain", "Inicio"));

      reportForm.addElement(grid);
    }
      catch (Exception e) { e.printStackTrace (); }

    page.append(reportForm.toString());

    page.append("</body></html>");

    return page.toString();
  }


  private String showHtmlForRegistering ()
  {
    String title = "Registrar mi número de matrícula";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));
```

```
    page.append("<h1>" + title + "</h1>");

      // Cree el objeto formulario HTML.
    HTMLForm registrationForm = new HTMLForm("LightsOn");

      // Defina un diseño de panel de dos columnas, en el que
      // deban disponerse los elementos HTML generados.
    GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

  try {
      // Configúrelo de tal manera que se llame a doPost() cuando se someta
el formulario.
      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("Número de matrícula:"));
      grid.addElement(licenseNum);

      grid.addElement(new LabelFormElement("Dirección de notificación por
correo-e:"));
      grid.addElement(eMailAddress);

      grid.addElement(new SubmitFormInput("submittingRegistration",
"Registrar"));
      grid.addElement(new SubmitFormInput("returningToMain", "Inicio"));

      registrationForm.addElement(grid);
    }
      catch (Exception e) { e.printStackTrace (); }

    page.append(registrationForm.toString());

    page.append("</body></html>");

    return page.toString();
  }


  private String showHtmlForUnregistering ()
  {
    String title = "Desregistrar mi número de matrícula";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));

    page.append("<h1>" + title + "</h1>");

      // Cree el objeto formulario HTML.
    HTMLForm unregistrationForm = new HTMLForm("LightsOn");
    GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

  try {
      // Configúrelo de tal manera que se llame a doPost() cuando se someta
el formulario.
```

```
      unregistrationForm.setMethod(HTMLForm.METHOD_POST);

      // Cree el objeto LineLayoutFormPanel.
      TextFormInput licenseNum = new TextFormInput("licenseNum");
      licenseNum.setSize(10);
      licenseNum.setMaxLength(10);

      grid.addElement(new LabelFormElement("Número de matrícula de
vehículo:"));
      grid.addElement(licenseNum);

      grid.addElement(new SubmitFormInput("submittingUnregistration",
"Desregistrar"));
      grid.addElement(new SubmitFormInput("returningToMain", "Inicio"));

      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 = "Todas las matrículas registradas";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));

                            try
    {
      // Cree el objeto formulario HTML.
      HTMLForm mainForm = new HTMLForm("LightsOn");

      // Defina un diseño de panel de una sola columna, en el que
      // deban disponerse los elementos HTML generados.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

      // Especifique el diseño para la tabla generada.
         HTMLTable table = new HTMLTable();
      table.setAlignment(HTMLConstants.LEFT);
      table.setBorderWidth(3);

      // Cree y añada el pie y la cabecera de tabla.
      HTMLTableCaption caption = new HTMLTableCaption();
      caption.setAlignment(HTMLConstants.TOP);
      caption.setElement(title);
      table.setCaption(caption);
      table.setHeader(new String[] { "Matrícula", "Fecha de adición" } );
```

```
      // Cree el conversor que generará la tabla HTML a partir del
      // conjunto de resultados devueltos por la consulta de base de datos.
   HTMLTableConverter converter = new HTMLTableConverter();
   converter.setTable(table);

      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Primero haga una preconsulta a la base de datos para verificar que
no está vacía.
      String query = "SELECT COUNT(*) FROM LICENSES";
                ResultSet rs = sqlStatement.executeQuery (query);
      rs.next();  // sitúe el cursor en la primera fila.
      int rowCount = rs.getInt(1);

      if (rowCount == 0) {
        page.append ("<font size=4 color=red>No se ha informado de ningún
vehículo.</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();
      // Nota: no se debe cerrar la sentencia mientras no se haya terminado
de usar el conjunto de resultados.

      grid.addElement(new SubmitFormInput("returningToMain", "Inicio"));

      mainForm.addElement(grid);
    page.append(mainForm.toString());
    }
      catch (Exception e) { e.printStackTrace (); }
    page.append("</body></html>");
    return page.toString();
  }


  private String showHtmlForListingAllReported ()
  {
    String title = "Todos los vehículos con las luces encendidas";
    StringBuffer page = new StringBuffer();
    page.append (showHeader (title));

                            try
    {
      // Cree el objeto formulario HTML.
      HTMLForm form = new HTMLForm("LightsOn");

      // Defina un diseño de panel de una sola columna, en el que
      // deban disponerse los elementos HTML generados.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

      // Especifique el diseño para la tabla generada.
         HTMLTable table = new HTMLTable();
      table.setAlignment(HTMLConstants.LEFT);
      table.setBorderWidth(3);
```

```
      // Cree y añada el pie y la cabecera de tabla.
      HTMLTableCaption caption = new HTMLTableCaption();
      caption.setAlignment(HTMLConstants.TOP);
      caption.setElement(title);
      table.setCaption(caption);
      table.setHeader(new String[] { "Matrícula", "Color", "Categoría",
"Fecha", "Hora" } );

      // Cree el conversor que generará la tabla HTML a partir del
      // conjunto de resultados devueltos por la consulta de base de datos.
   HTMLTableConverter converter = new HTMLTableConverter();
   converter.setTable(table);

      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Primero haga una preconsulta a la base de datos para verificar que
no está vacía.
      String query = "SELECT COUNT(*) FROM REPORTS";
                ResultSet rs = sqlStatement.executeQuery (query);
      rs.next();  // sitúe el cursor en la primera fila.
      int rowCount = rs.getInt(1);

      if (rowCount == 0) {
        page.append ("<font size=4 color=red>No se ha informado de ningún
vehículo.</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();
      // Nota: no se debe cerrar la sentencia mientras no se haya terminado
de usar el conjunto de resultados.

      grid.addElement(new SubmitFormInput("returningToMain", "Inicio"));
      form.addElement(grid);
      page.append(form.toString());
    }
      catch (Exception e) { e.printStackTrace (); }
    page.append("</body></html>");
    return page.toString();
  }

}
```

»Cómo escribir el primer programa de Toolbox para Java

Para empezar este sencillo ejercicio, debe tener instalado Java en la estación de trabajo. Puede decidir qué versión desea instalar consultando los requisitos para ejecutar aplicaciones Java.

Una vez que tenga instalado Java en el cliente, lleve a cabo las tareas siguientes:

1. Copie jt400.jar en la estación de trabajo.
2. Añada jt400.jar a la CLASSPATH de la estación de trabajo especificando la vía de acceso completa del archivo JAR a la CLASSPATH. Por ejemplo, si el archivo jt400.jar reside en el directorio c:\lib de la estación de trabajo (que ejecuta Windows), añada lo siguiente al final de la sentencia CLASSPATH:

   ```
   ;c:\lib\jt400.jar
   ```
3. Abra un editor de texto y escriba el primer ejemplo simple de programación.

   Nota: asegúrese de no incluir el texto que hace referencia a las notas (por ejemplo, Nota 1, Nota 2, etc.). Guarde el documento nuevo con el nombre CmdCall.java.
4. Inicie una sesión de mandato en la estación de trabajo y utilice el siguiente mandato para compilar el ejemplo simple de programación:

   ```
   javac CmdCall.java
   ```
5. En la sesión de mandato, escriba el siguiente mandato para ejecutar el ejemplo simple de programación:

   ```
   java CmdCall
   ```

«

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Ejemplos simples de programación]

Ejemplo: cómo se utiliza CommandCall

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de cómo se utiliza la clase de acceso de IBM Toolbox para Java,
CommandCall.
//
// Este fuente es un ejemplo de "lista de trabajos" de IBM Toolbox para
Java.
//
////////////////////////////////////////////////////////////////////////////
//
// Las clases de acceso de la Caja de Herramientas están en el paquete
com.ibm.as400.access.
// Importe este paquete para utilizar las clases de la Caja de Herramientas.
//
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class CmdCall
{
    public static void main (String[] args)
   {
      // Como sucede con otras clases Java, las clases de la Caja de
Herramientas
      // lanzan excepciones cuando algo falla. Los programas que utilizan la
      // la Caja de Herramientas deben capturar estas excepciones.
      tryNota 1
      {
AS400 system = new AS400();

         CommandCall cc = new CommandCall(system);Nota 2


         cc.run("CRTLIB MYLIB");Nota 3


         AS400Message[] ml = cc.getMessageList();Nota 4

         for (int i=0; i<ml.length; i++)
         {
```

```
            System.out.println(ml[i].getText());Nota 5
          }
        }
          catch (Exception e)
        {
        e.printStackTrace();
        }

         System.exit(0);
    }
}
```

1. Toolbox para Java utiliza el objeto "AS400" para identificar el servidor destino. Si construye el objeto AS400 sin parámetros, Toolbox para Java solicita el nombre de sistema, el ID de usuario y la contraseña. La clase AS400 también incluye un constructor que toma el nombre de sistema, el ID de usuario y la contraseña.

2. Utilice el objeto CommandCall de Toolbox para Java para enviar mandatos al servidor. Al crear el objeto CommandCall, le pasa un objeto AS400 para que sepa cuál es el servidor destino del mandato.

3. Utilice el método run() en el objeto de llamada a mandato para ejecutar un mandato.

4. El resultado de ejecutar un mandato es una lista de mensajes de OS/400. La Caja de Herramientas representa estos mensajes como objetos AS400Message. Una vez completado el mandato, el usuario recibe los mensajes obtenidos del objeto CommandCall.

5. Imprima el texto del mensaje. También hay disponible otra información como el ID de mensaje, la gravedad del mensaje, etc. Este programa sólo imprime el texto del mensaje.

[Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Parte siguiente | Ejemplos simples de programación]

Ejemplo: cómo se utilizan las colas de mensajes (parte 1 de 3)

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de cómo se utiliza la función "cola de mensajes" de IBM Toolbox
para Java
//
// Este fuente es un ejemplo de la "cola de mensajes" de IBM Toolbox para
Java.
//
////////////////////////////////////////////////////////////////////////////

package examples; Nota 1


import java.io.*;
import java.util.*;

import com.ibm.as400.access.*; Nota 2

public class displayMessages extends Object
{

   public static void main(String[] parameters) Nota 3
   {
      displayMessages me = new displayMessages();
      me.Main(parameters); Nota 4


      System.exit(0); Nota 5
   }


   void displayMessage()
   {
   }


   void Main(String[] parms)
```

```
    {
       try Nota 6
       {

                // El código de IBM Toolbox para Java va aquí
       }
         catch (Exception e)
       {
          e.printStackTrace(); Nota 7
       }
    }
}
```

1. Esta clase está en el paquete 'examples'. Java utiliza paquetes para evitar que se produzcan conflictos de nombres entre los archivos de clase Java.

2. Esta línea pone a disposición de este programa todas las clases de IBM Toolbox para Java existentes en el paquete access. Estas clases tienen en común el prefijo **com.ibm.as400**. Mediante una sentencia de importación (import), el programa puede hacer referencia a una clase tan solo con utilizar su nombre, no el nombre totalmente calificado. Por ejemplo, para hacer referencia a la clase AS400 basta con utilizar el nombre AS400, en vez de com.ibm.as400.AS400.

3. Esta clase tiene un método **main**; por lo tanto, se puede ejecutar como una aplicación. Para invocar el programa, ejecute **java examples.displayMessages**. Observe que las mayúsculas/minúsculas deben coincidir al ejecutar el programa. Debido a que se utiliza una clase de IBM Toolbox para Java, el archivo jt400.zip debe estar en la variable de entorno de vía de acceso de clases (classpath).

4. El método main indicado en la nota 3 es estático. Una de las restricciones que tienen los métodos estáticos es que solo pueden llamar a otros métodos estáticos de sus propias clases. Para evitar esta restricción, muchos programas Java crean un objeto y luego realizan el proceso de inicialización en un método llamado **Main**. El método Main() puede llamar a cualquier otro método del objeto displayMessages.

5. IBM Toolbox para Java crea hebras en nombre de la aplicación para llevar a cabo la actividad de IBM Toolbox para Java. El programa, si cuando va a terminar no emite **System.exit(0)**, no puede terminar con normalidad. Por ejemplo, supongamos que este programa se ha ejecutado desde un indicador de solicitud del DOS de Windows 95. Sin esta línea, el indicador de solicitud de mandatos no retornaría cuando acabase el programa. El usuario tendría que pulsar las teclas Control-C para obtener un indicador de solicitud de mandatos.

6. El código de IBM Toolbox para Java lanza excepciones que el programa Java debe capturar.

7. Este programa visualiza el texto de la excepción mientras lleva a cabo el proceso de error. Las excepciones lanzadas por IBM Toolbox para Java están traducidas, por lo que el texto de la excepción estará en el idioma de la estación de trabajo.

[Parte siguiente | Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Parte anterior | Parte siguiente | Ejemplos simples de programación]

Ejemplo: cómo se utilizan las colas de mensajes (parte 2 de 3)

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de cómo se utiliza la función "cola de mensajes" de IBM Toolbox
para Java
//
// Este fuente es un ejemplo de la "cola de mensajes" de IBM Toolbox para
Java.
//
////////////////////////////////////////////////////////////////////////////

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(); Nota 1


           if (parms.length > 0)
                          system.setSystemName(parms[0]); Nota 2

        }
          catch (Exception e)
        {
        e.printStackTrace();
        }
    }
}
```

1. Un programa utiliza el objeto **AS400** para designar a qué servidor debe conectarse. Todos los programas que necesitan recursos de un servidor deben tener un objeto AS400, con una sola salvedad. La excepción es JDBC. Si el programa utiliza JDBC, el controlador JDBC de IBM Toolbox para Java crea el objeto AS400 para el programa.

2. Este programa supone que el primer parámetro de línea de mandatos es el nombre del servidor. Si se pasa un parámetro al programa, se utiliza el método **setSystemName** del objeto AS400 para establecer el nombre del sistema. El objeto AS400 también necesita información de inicio de sesión del servidor:
    - Si el programa se está ejecutando en una estación de trabajo, el programa de IBM Toolbox para Java solicita al usuario un ID de usuario y una contraseña. **Nota:** si no se especifica un nombre del sistema como parámetro de línea de mandatos, el objeto AS400 también solicita el nombre del sistema.
    - Si el programa se está ejecutando en la máquina virtual Java de iSeries, se utiliza el ID de usuario y la contraseña del usuario que ejecuta el programa Java. En este caso, el usuario no especifica un nombre del sistema, sino que deja que el nombre del sistema tome por omisión el nombre del sistema en el que se está ejecutando el programa.

[Parte anterior | Parte siguiente | Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Parte anterior | Ejemplos simples de programación]

Ejemplo: cómo se utilizan las colas de mensajes (parte 3 de 3)

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo de cómo se utiliza la función "cola de mensajes" de IBM Toolbox
para Java
//
// Este fuente es un ejemplo de la "cola de mensajes" de IBM Toolbox para
Java.
//
///////////////////////////////////////////////////////////////////////////

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); Nota 1


                  Enumeration e = queue.getMessages(); Nota 2

      while (e.hasMoreElements())
                  {

                      QueuedMessage message = (QueuedMessage)
e.nextElement(); Nota 3

                      System.out.println(message.getText()); Nota 4
                   }
               }
        catch (Exception e)
               {
      e.printStackTrace();
               }
         }
}
```

1. La finalidad de este programa es visualizar mensajes en una cola de mensajes de servidor. Para esta tarea se utiliza el objeto **MessageQueue** de IBM Toolbox para Java. Cuando se construye el objeto cola de mensajes, los parámetros son el objeto AS400 y el nombre de la cola de mensajes. El objeto AS400 indica qué servidor contiene el recurso y el nombre de la cola de mensajes identifica la cola de mensajes del servidor. En este caso, se utiliza una constante que indica al objeto cola de mensajes que acceda a la cola del usuario que ha iniciado la sesión.

2. El objeto cola de mensajes obtiene una lista de mensajes del servidor. En este momento se establece la conexión con el servidor.

3. Eliminar un mensaje de la lista. El mensaje está en el objeto QueuedMessage del programa de IBM Toolbox para Java.

4. Imprimir el texto del mensaje.

[Parte anterior | Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Parte siguiente | Ejemplos simples de programación]

Ejemplo: cómo se utiliza el acceso a nivel de registro (parte 1 de 2)

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de acceso a nivel de registro (RLA). Este programa solicitará
// el nombre del servidor y el archivo que se debe visualizar. El
// archivo debe existir y ha de contener registros. Cada registro del
archivo
// se visualizará en la salida del sistema (System.out).
//
// Sintaxis de la llamada: java RLSequentialAccessExample
//
// Este fuente es un ejemplo de "RecordLevelAccess" de IBM Toolbox para
Java.
//
////////////////////////////////////////////////////////////////////////////

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("Nombre del sistema: ");
        systemName = inputStream.readLine();

        System.out.print("Biblioteca en la que existe el archivo: ");
        library = inputStream.readLine();
```

```
        System.out.print("Nombre de archivo: ");
        file = inputStream.readLine();

        System.out.print("Nombre de miembro (pulse Intro para el primer
miembro): ");
        member = inputStream.readLine();
        if (member.equals(""))
        {
          member = "*FIRST";
        }

    System.out.println();
      }
        catch (Exception e)
      {
        System.out.println("Error al obtener entrada de usuario.");
      e.printStackTrace();
       System.exit(0);
      }


      AS400 system = new AS400(systemName); Nota 1
                            try
      {
        system.connectService(AS400.RECORDACCESS);
      }
      catch(Exception e)
      {
      System.out.println("No ha sido posible conectar para acceso a nivel de
registro.");
        System.out.println("Vea si en el archivo de instalación de la guía
del programador hay instrucciones especiales relacionadas con el acceso a
nivel de registro");
      e.printStackTrace();
       System.exit(0);
      }


      QSYSObjectPathName filePathName = new QSYSObjectPathName(library,
file, member, "MBR"); Nota 2


      SequentialFile theFile = new SequentialFile(system,
filePathName.getPath()); Nota 3


      AS400FileRecordDescription recordDescription = new
AS400FileRecordDescription(system, filePathName.getPath());
                            try
      {
        RecordFormat[] format = recordDescription.retrieveRecordFormat();
Nota 4


        theFile.setRecordFormat(format[0]); Nota 5


        theFile.open(AS400File.READ_ONLY, 100,
```

```
AS400File.COMMIT_LOCK_LEVEL_NONE); Nota 6

        System.out.println("Se va a visualizar el archivo " +
library.toUpperCase() + "/" + file.toUpperCase() + "(" +
theFile.getMemberName().trim() + "):");


        Record record = theFile.readNext(); Nota 7
     while(record != null)
        {
          System.out.println(record);
          record = theFile.readNext();
        }
    System.out.println();


        theFile.close(); Nota 8


        system.disconnectService(AS400.RECORDACCESS); Nota 9
      }
        catch (Exception e)
      {
        System.out.println("Se produjo un error al intentar visualizar el
archivo.");
      e.printStackTrace();

                            try
        {
         // Cierre el archivo.
        theFile.close();
        }
        catch(Exception x)
        {
        }

    system.disconnectService(AS400.RECORDACCESS);
       System.exit(0);
      }

      // Asegúrese de que finaliza la aplicación; en el readme encontrará
detalles
       System.exit(0);
    }
  }
```

1. Esta línea de código crea un objeto AS400 y establece conexión con el servicio de acceso a nivel de registro.
2. Esta línea crea un objeto QSYSObjectPathName que obtiene el formato del nombre de vía de sistema de archivos integrado del objeto que se debe visualizar.
3. Esta sentencia crea un objeto que representa un archivo secuencial existente en el servidor con el que se ha establecido conexión. Este archivo secuencial es el que se visualizará.
4. Estas líneas recuperan el formato de registro del archivo.
5. Esta línea establece el formato de registro del archivo.
6. Esta línea abre para lectura el archivo seleccionado. Leerá los registros de 100 en 100, cuando sea posible.

7. Esta línea de código lee cada registro por orden.
8. Esta línea cierra el archivo.
9. Esta línea desconecta del servicio de acceso a nivel de registro.

[Parte siguiente | Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Parte anterior | Ejemplos simples de programación]

Ejemplo: cómo se utiliza el acceso a nivel de registro (parte 2 de 2)

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de acceso a nivel de registro.
//
// Sintaxis de llamada: 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"; Nota 1


                          try
    {
      SequentialFile theFile = new SequentialFile(system, filePathName);

      // Inicio de nota 2
      CharacterFieldDescription lastNameField = new
CharacterFieldDescription(new AS400Text(20), "APELLIDO");
      CharacterFieldDescription firstNameField = new
CharacterFieldDescription(new AS400Text(20), "NOMBRE");
      BinaryFieldDescription yearsOld = new BinaryFieldDescription(new
AS400Bin4(), "EDAD");

      RecordFormat fileFormat = new RecordFormat("RF");
      fileFormat.addFieldDescription(lastNameField);
      fileFormat.addFieldDescription(firstNameField);
      fileFormat.addFieldDescription(yearsOld);

      theFile.create(fileFormat, "Un archivo de nombres y edades"); Nota 2

```

```
      // Fin de nota 2


      theFile.open(AS400File.READ_WRITE, 1,
AS400File.COMMIT_LOCK_LEVEL_NONE);

      // Inicio de nota 3
      Record newData = fileFormat.getNewRecord();
      newData.setField("APELLIDO", "Doe");
      newData.setField("NOMBRE", "John");
      newData.setField("EDAD", new Integer(63));


      theFile.write(newData); Nota 3
      // Fin de nota 3

        theFile.close();
    }
    catch(Exception e)
    {
      System.out.println("Se ha producido un error: ");
      e.printStackTrace();
    }

    system.disconnectService(AS400.RECORDACCESS);

       System.exit(0);
  }
}
```

1. (args[0]) en la línea anterior y MYFILE.FILE son fragmentos de código necesarios para que se ejecute el resto del ejemplo. El programa supone que existe la biblioteca MYLIB en el servidor y que el usuario puede acceder a ella.

2. El texto situado entre los comentarios Java que tienen la etiqueta "Inicio de nota 2" y "Fin de nota 2" muestra cómo puede crear usted mismo un formato de registro en vez de obtenerlo de un archivo existente. La última línea de este bloque crea el archivo en el servidor.

3. El texto situado entre los comentarios Java que tienen la etiqueta "Inicio de nota 3" y "Fin de nota 3" muestra una forma de crear un registro y después escribirlo en un archivo.

[Parte anterior | Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Parte siguiente | Ejemplos simples de programación]

Ejemplo: cómo se utilizan las clases JDBC para crear y llenar con datos una tabla (parte 1 de 2)

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////////
//
// Ejemplo JDBCPopulate. Este programa utiliza el controlador JDBC de
// IBM Toolbox para Java a fin de crear una tabla y llenarla con datos.
//
// Sintaxis de mandato:
//    JDBCPopulate sistema nombreColección nombreTabla
//
// Por ejemplo:
//    JDBCPopulate MiSistema MiBiblioteca MiTabla
//
// Este fuente es un ejemplo del controlador JDBC de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{

    private static final String words[]
        = { "Uno",      "Dos",      "Tres",    "Cuatro",     "Cinco",
            "Seis",      "Siete",    "Ocho",    "Nueve",     "Diez",
            "Once",   "Doce",   "Trece", "Catorce", "Quince",
            "Dieciséis",  "Diecisiete","Dieciocho", "Diecinueve", "Veinte"
};



    public static void main (String[] parameters)
    {

        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Utilización:");
            System.out.println("");
            System.out.println("   JDBCPopulate sistema nombreColección
nombreTabla");
            System.out.println("");
            System.out.println("");
```

```
            System.out.println("Por ejemplo:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCPopulate MiSistema MiBiblioteca
MiTabla");
            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()); Nota 1


            connection = DriverManager.getConnection ("jdbc:as400://"
                + system + "/" + collectionName); Nota 2


  try {
                Statement dropTable = connection.createStatement ();
                dropTable.executeUpdate ("DROP TABLE " + tableName); Nota 3

            }
            catch (SQLException e) {
            }


            Statement createTable = connection.createStatement ();
            createTable.executeUpdate ("CREATE TABLE " + tableName
                + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
                + " SQUAREROOT DOUBLE)"); Nota 4


            PreparedStatement insert = connection.prepareStatement ("INSERT
INTO "
                + tableName + " (I, WORD, SQUARE, SQUAREROOT) "
                + " VALUES (?, ?, ?, ?)"); Nota 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 (); Nota 6
            }

            System.out.println ("La tabla " + collectionName + "." +
tableName
```

```
                + " se ha llenado con datos.");
        }

    catch (Exception e) {
         System.out.println ();
            System.out.println ("ERROR: " + e.getMessage());
        }

        finally {

  try {
            if (connection != null)

                    connection.close (); Nota 7
            }
            catch (SQLException e) {
                // Hacer caso omiso.
            }
        }

        System.exit (0);
    }



}
```

1. Esta línea carga el controlador JDBC de IBM Toolbox para Java. Se necesita un controlador JDBC para que haga de mediador entre JDBC y la base de datos con la que se trabaja.

2. Esta sentencia conecta con la base de datos. Aparecerá una solicitud para el ID de usuario y la contraseña. Se proporciona un esquema por omisión para que no tenga usted que calificar el nombre de la tabla en las sentencias SQL.

3. Estas líneas suprimen la tabla, si es que ya existe.

4. Estas líneas crean la tabla.

5. Esta línea prepara una sentencia que insertará filas en la tabla. Debido a que se va a ejecutar esta sentencia varias veces, debe utilizarse una sentencia preparada (PreparedStatement) y marcadores de parámetro.

6. Este bloque de código llena con datos la tabla; cada vez que se ejecuta el bucle, el bloque inserta una fila en la tabla.

7. Ahora que ya se ha creado la tabla y se ha llenado con datos, esta sentencia cierra la conexión con la base de datos.

[Parte siguiente | Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

Ejemplo: cómo se utilizan las clases JDBC para crear y llenar con datos una tabla (parte 2 de 2)

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo JDBCQuery. Este programa utiliza el controlador JDBC de IBM para
// consultar una tabla y enviar su contenido a la salida.
//
// Sintaxis de mandato:
//    JDBCQuery sistema nombreColección nombreTabla
//
// Por ejemplo:
//    JDBCQuery MiSistema qiws qcustcdt
//
// Este fuente es un ejemplo del controlador JDBC de IBM Toolbox para Java.
//
////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{



    // Dé formato a una serie (String) para que tenga la anchura
especificada.
    private static String format (String s, int width)
    {
        String formattedString;

        // La serie es más corta que la anchura especificada,
        // por lo que tenemos que rellenarla con blancos.
        if (s.length() < width) {
            StringBuffer buffer = new StringBuffer (s);
            for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
            formattedString = buffer.toString();
        }

        // En el caso contrario, tendremos que truncar la serie.
          else
            formattedString = s.substring (0, width);
```

```
        return formattedString;
    }



    public static void main (String[] parameters)
    {
        // Compruebe los parámetros de entrada.
        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Utilización:");
            System.out.println("");
            System.out.println("   JDBCQuery sistema nombreColección
nombreTabla");
            System.out.println("");
            System.out.println("");
            System.out.println("Por ejemplo:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCQuery MiSistema 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()); Nota 1


            // Obtenga una conexión con la base de datos. Debido a que no
            // proporcionamos un ID de usuario ni una contraseña, aparecerá
una solicitud.            connection = DriverManager.getConnection
("jdbc:as400://" + system);
            DatabaseMetaData dmd = connection.getMetaData (); Nota 2


            // Ejecute la consulta.            Statement select =
connection.createStatement ();
            ResultSet rs = select.executeQuery ("SELECT * FROM "
                + collectionName + dmd.getCatalogSeparator() + tableName);
Nota 3


            // Obtenga información sobre el conjunto de resultados.
Establezca que la anchura
            // de la columna sea la longitud mayor de las dos: la longitud
de la etiqueta
            // o la longitud de los datos.            ResultSetMetaData rsmd
= rs.getMetaData ();
            int columnCount = rsmd.getColumnCount (); Nota 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)); Nota 5
            }

            // Envíe a la salida las cabeceras de columna.
                for (int i = 1; i <= columnCount; ++i) {
                System.out.print (format (rsmd.getColumnLabel(i),
columnWidths[i-1]));
                    System.out.print (" ");
            }
         System.out.println ();

            // Envíe a la salida una línea de guiones.
            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 ();

            // Itere a través de las filas del conjunto de resultados y
envíe a la
            // salida las columnas que hay en cada fila.             while
(rs.next ()) {
                for (int i = 1; i <= columnCount; ++i) {
                    String value = rs.getString (i);
                    if (rs.wasNull ())
                        value = "<null>"; Nota 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 {

            // Borre.
  try {
            if (connection != null)
               connection.close ();
            }
            catch (SQLException e) {
                // Hacer caso omiso.
            }
        }

        System.exit (0);
    }
```

```
}
```

1. Esta línea carga el controlador JDBC de IBM Toolbox para Java. El controlador JDBC hace de mediador entre JDBC y la base de datos con la que está trabajando.

2. Esta línea recupera los metadatos de la conexión; los metadatos son un objeto que describe muchas de las características de la base de datos.

3. Esta sentencia ejecuta la consulta en la tabla especificada.

4. Estas líneas recuperan información acerca de la tabla.

5. Estas líneas establecen la anchura de la columna para que sea igual a la longitud mayor de estas dos: la longitud de la etiqueta o la longitud de los datos.

6. Este bloque de código itera a través de todas las filas de la tabla y visualiza el contenido de cada columna de cada fila.

[Parte anterior | Ejemplos simples de programación]

Para ver una descripción detallada en una ventana aparte, el navegador debe tener habilitado el soporte para JavaScript. Si el navegador no tiene habilitado JavaScript o no soporta JavaScript, pulse el enlace de texto **Nota** o desplácese al final de la página para ver la explicación.

[Ejemplos simples de programación]

Ejemplo: visualizar una lista de trabajos servidores en una GUI

Puede utilizar el siguiente código como ejemplo para su programa. El ejemplo contiene descripciones detalladas sobre las líneas clave del código. Puede ver las descripciones detalladas de los modos siguientes:

- Pulse la imagen  para ver la explicación detallada en una ventana emergente.
- Pulse el enlace de texto para ver la explicación detallada que se proporciona al final del ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de cómo se utiliza la clase de vaccess de IBM Toolbox para Java,
// VJobList.
//
// Este fuente es un ejemplo de "lista de trabajos" de IBM Toolbox para
Java.
//
////////////////////////////////////////////////////////////////////////////

package examples; Nota 1 

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*; Nota 2 

import javax.swing.*; Nota 3 
import java.awt.*;
import java.awt.event.*;

public class GUIExample
{

  public static void main(String[] parameters) Nota 4 
  {
     GUIExample example = new GUIExample(parameters);

  }

  public GUIExample(String[] parameters)
  {
     try Nota 5 
     {
       // Cree un objeto AS400.
     //Se ha pasado el nombre del sistema como primer argumento de línea de
mandatos.
```

```
          AS400 system = new AS400 (parameters[0]); Nota 6


          VJobList jobList = new VJobList (system); Nota 7

            // Cree un marco.
     JFrame frame = new JFrame ("Ejemplo de lista de trabajos"); Nota 8

     // Cree un adaptador de diálogo de errores. Así los errores se
visualizarán al usuario.
     ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (frame); Nota
9


    // Cree una sección de panel explorador para presentar la lista de
trabajos.     AS400ExplorerPane explorerPane = new AS400ExplorerPane
(jobList); Nota 10


     explorerPane.addErrorListener (errorHandler); Nota 11

            // Utilice load para cargar la información desde el sistema.
     explorerPane.load(); Nota 12

     // Cuando el marco se cierre, salga del programa.
     frame.addWindowListener (new WindowAdapter () Nota 13
     {
                public void windowClosing (WindowEvent event)
          {
       System.exit(0);
          }
     } );

            // Diseñe el marco con la sección de panel explorador.
    frame.getContentPane().setLayout(new BorderLayout() );
frame.getContentPane().add("Center", explorerPane); Nota 14


   frame.pack();
     frame.show(); Nota 15
     }

        catch (Exception e)
     {
          e.printStackTrace(); Nota 16


     System.exit(0); Nota 17
     }
     }
}
```

1. Esta clase está en el paquete de ejemplos (examples). Java utiliza paquetes para evitar que se produzcan conflictos de nombres entre los archivos de clase Java.

2. Esta línea pone a disposición de este programa todas las clases de IBM Toolbox para Java existentes en el paquete vaccess. Estas clases tienen en común el prefijo com.ibm.as400.vaccess. Mediante una sentencia de importación

(import), el programa llama al nombre, en vez de al paquete más el nombre. Por ejemplo, para hacer referencia a la clase AS400ExplorerPane basta con utilizar AS400ExplorerPane, en vez de com.ibm.as400.AS400ExplorerPane.

3. Esta línea hace que todas las clases JFC (clases Java fundamentales) del paquete Swing estén disponibles para este programa. Los programas Java que utilizan las clases vaccess (GUI) de IBM Toolbox para Java necesitan el JDK 1.1.2 más Java Swing 1.0.3 de Sun Microsystems, Inc. Swing está disponible en las JFC 1.1 de Sun.

4. Esta clase tiene un método main por lo que puede ejecutarse como una aplicación. Para invocar el programa, ejecute "java examples.GUIExample nombreServidor", siendo nombreServidor el nombre del servidor. Para que esto funcione, el archivo jt400.zip o el archivo jt400.jar debe estar en su vía de acceso de clases (classpath).

5. El código de IBM Toolbox para Java lanza excepciones que el programa debe capturar.

6. IBM Toolbox para Java utiliza la clase AS400. Esta clase gestiona información de inicio de sesión, crea y mantiene conexiones por socket, y envía y recibe datos. En este ejemplo, el programa pasará el nombre del servidor al objeto AS400.

7. La clase VJobList permite a IBM Toolbox para Java representar una lista de trabajos servidores que se pueden visualizar en un componente de vaccess (GUI). Observe que se utiliza el objeto AS400 para especificar el servidor en el que reside la lista.

8. Esta línea construye un marco o una ventana de nivel superior que se utilizará para visualizar la lista de trabajos.

9. ErrorDialogAdapter es un componente GUI (interfaz gráfica de usuario) de IBM Toolbox para Java que se crea para visualizar automáticamente una ventana de diálogo cada vez que se produce un evento de error en la aplicación.

10. Esta línea crea una sección de panel explorador de AS/400 (AS400ExplorerPane), que es una interfaz gráfica de usuario (GUI) que representa una jerarquía de objetos dentro de un recurso de servidor. La sección AS400ExplorerPane presenta, a la izquierda, un árbol enraizado en la lista de trabajos VJobList y, a la derecha, los detalles del recurso. Esta línea solo inicializa la sección en un estado por omisión y no carga el contenido de la lista de trabajos VJobList en la sección.

11. Esta línea añade el manejador de errores creado en el paso 9 como escuchador del componente GUI (interfaz gráfica de usuario) VJobList.

12. Esta línea carga el contenido de la lista de trabajos (JobList) en la sección de panel explorador (ExplorerPane). Es preciso llamar explícitamente a este método para establecer comunicación con el servidor y cargar información del mismo. Esto transfiere el control a la aplicación cuando se establece comunicación con el servidor. Con esta línea puede:
    - Cargar el contenido antes de añadir la sección a un marco. El marco no aparece hasta que se ha cargado toda la información, como en este ejemplo.
    - Cargar el contenido después de añadir la sección a un marco y de visualizar el marco. El marco aparece con un "cursor de espera" y la información se va rellenando a medida que se carga.

13. Esta línea añade un escuchador de ventana para que la aplicación finalice cuando se cierre el marco.

14. Esta línea añade el componente GUI (interfaz gráfica de usuario) de lista de trabajos al centro del marco de control.

15. Esta línea llama al método show para que la ventana se establezca como visible para el usuario.

16. Las excepciones de IBM Toolbox para Java están traducidas para que el texto aparezca en el idioma de la estación de trabajo. Por ejemplo, este programa visualiza el texto de la excepción como su propio proceso de error.

17. IBM Toolbox para Java crea hebras para llevar a cabo la actividad de IBM Toolbox para Java. El programa, si no hace System.exit(0) cuando termina, no puede salir (exit) con normalidad. Por ejemplo, si el programa se ejecutase desde un indicador de solicitud del DOS de Windows 95 sin esta línea, el indicador de solicitud de mandatos no retornaría cuando acabase el programa.

[Ejemplos simples de programación]

Ejemplos: consejos para la programación

En esta sección figura una lista de los ejemplos de código proporcionados en toda la documentación del tema dedicado a la gestión de conexiones.

Gestión de conexiones

- Ejemplo: establecer una conexión con el servidor iSeries con un objeto CommandCall
- Ejemplo: establecer dos conexiones con el servidor iSeries con un objeto CommandCall
- Ejemplo: crear los objetos CommandCall y IFSFileInputStream con un objeto AS400
- Ejemplo: cómo se utiliza AS400ConnectionPool para preconectar con el servidor iSeries
- Ejemplo: cómo se utiliza AS400ConnectionPool para preconectar con un servicio específico en el servidor iSeries y después reutilizar la conexión

Inicio y finalización de las conexiones

- Ejemplo: cómo se preconecta un programa Java al servidor iSeries
- Ejemplo: cómo se desconecta un programa Java de un servidor iSeries
- Ejemplo: cómo se desconecta y reconecta un programa Java al servidor iSeries con disconnectService() y run()
- Ejemplo: cómo se desconecta un programa Java del servidor iSeries y no se puede reconectar

Excepciones

- Ejemplo: cómo se utilizan las excepciones

Eventos de error

- Ejemplo: manejar eventos de error
- Ejemplo: definir un escuchador de errores
- Ejemplo: cómo se utiliza un manejador personalizado para manejar eventos de error

Rastreo

- Ejemplo: cómo se utiliza el rastreo
- Ejemplo: cómo se utiliza setTraceOn()
- Ejemplo: cómo se utiliza el rastreo de componente

Optimización

- Ejemplo: crear dos objetos AS400
- Ejemplo: cómo se utiliza un objeto AS400 para representar un segundo servidor

Instalación y actualización

- Ejemplo: cómo se utiliza la clase AS400ToolboxInstaller

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplos: ToolboxME para iSeries

En esta sección figura una lista de los ejemplos de código que se proporcionan en la documentación de ToolboxME para iSeries.

- Ejemplo: crear un ejemplo de ToolboxME para iSeries - JdbcDemo.java
- Ejemplo: cómo se utiliza ToolboxME para iSeries, MIDP y JDBC
- Ejemplo: cómo se utiliza ToolboxME para iSeries, MIDP y Toolbox para Java

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

»

Ejemplo: cómo se utiliza ToolboxME para iSeries, MIDP y JDBC

El fuente siguiente muestra un modo en que la aplicación de ToolboxME para iSeries puede utilizar MIDP (Mobile Information Device Profile) y JDBC para acceder a una base de datos y almacenar información fuera de línea.

En este ejemplo se muestra cómo un agente de la propiedad inmobiliaria puede ver las propiedades que están a la venta en ese momento y ofertar por ellas. El agente utiliza un dispositivo Tier0 para acceder a la información acerca de las propiedades, que se almacena en la base de datos del servidor iSeries.

Una vez compilado como un programa de trabajo, el código de ejemplo siguiente conecta a una base de datos creada a este efecto.

Para crear una versión de trabajo del código fuente y obtener el fuente para crear y llenar con datos la base de datos necesaria, debe bajar el ejemplo. Puede que también desee consultar las instrucciones para crear y ejecutar el programa de ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de ToolboxME para iSeries. Este programa es un MIDlet de ejemplo
que muestra cómo
// puede codificar una aplicación JdbcMe para el perfil MIDP. Consulte los
métodos
// startApp, pauseApp, destroyApp y commandAction para ver cómo maneja cada
// transición solicitada.
//
////////////////////////////////////////////////////////////////////////////

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 hay ofertas disponibles,
seleccione aquí para bajar ofertas";
    private List      main = new List("Demostración de ofertas de JdbcMe",
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("Salir", Command.SCREEN, 0);
    private Command backCommand   = new Command("Atrás", Command.SCREEN, 0);
    private Command cancelCommand = new Command("Cancelar", Command.SCREEN,
0);
    private Command goCommand     = new Command("Ir", Command.SCREEN, 1);
    private Displayable   onErrorGoBackTo = null;

    /*
     * Construya un nuevo objeto JdbcMidpBid.
     */
    public JdbcMidpBid()
    {
        display = Display.getDisplay(this);
    }

    /**
     * Visualice la pantalla principal.
     */
    public void startApp()
    {
        main.append("Mostrar ofertas", null);
        main.append("Obtener nuevas ofertas", null);
        main.append("Valores", null);
        main.append("Acerca de", null);
        main.addCommand(exitCommand);
        main.setCommandListener(this);

        display.setCurrent(main);
    }

    public void commandAction(Command c, Displayable s)
    {
        // Todo el proceso de exitCommand es el mismo.
        if (c == exitCommand)
        {
            destroyApp(false);
            notifyDestroyed();
            return;
        }
        if (s instanceof List)
        {
            List     current = (List)s;

            // Se ha producido una acción en la página principal.
            if (current == main)
            {
                int   idx = current.getSelectedIndex();
                switch (idx)
                {
                case 0:     // Mostrar ofertas actuales
                    showBids();
              break;
                case 1:     // Obtener nuevas ofertas
```

```
                    getNewBids();
              break;
                case 2:     // Valores
                    doSettings();
              break;
                case 3:     // Acerca de
                    aboutBox();
              break;
            default:
              break;
                }
             return;
             } // current == main

             // Se ha producido una acción en la página de listas
             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";
                     // Haga un seguimiento también de qué fila de conjunto
de resultados
                     // fuera de línea es. Coincide que es la misma que el
índice
                     // de la lista.
                     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)
                 {
                     // Se ha terminado de trabajar con los valores.
                     display.setCurrent(main);
```

```
                settingsForm = null;
            return;
                }
            } // settingsForm
            if (current == aboutBox)
            {
                if (c == backCommand)
                {
                    // Se ha terminado de trabajar con el recuadro Acerca
de.
                    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)
                    {
                        // Si ya no nos encontramos en
                        // bidForm, nos deshacemos de él.
                        bidForm = null;
                    }
            return;
                }
            return;
            } // current == bidForm
        } // instanceof Form
    }

    public void aboutBox()
    {
        aboutBox = new Form("aboutbox");
        aboutBox.setTitle("Acerca de");
        aboutBox.append(new StringItem("", "Ejemplo de agente inmobiliario
de Midp para JdbcMe "));
        aboutBox.addCommand(backCommand);
        aboutBox.setCommandListener(this);
        display.setCurrent(aboutBox);
    }

    /**
     *  El formulario de valores.
     */
    public void doSettings()
    {
        settingsForm = new Form("settingsform");
        settingsForm.setTitle("Valores");
        settingsForm.append(new StringItem("", "URL de base de datos"));
        settingsForm.append(urlText);
        settingsForm.append(new StringItem("", "Servidor JdbcMe"));
        settingsForm.append(jdbcmeText);
        settingsForm.append(new StringItem("", "Rastreo"));
```

```
        settingsForm.addCommand(backCommand);
        settingsForm.setCommandListener(this);
        display.setCurrent(settingsForm);
    }


    /**
     * Visualice la pantalla de oferta para el objetivo de la oferta
     * seleccionado.
     */
    public void bidOnProperty()
    {
        StringItem item = new StringItem("", bidTarget);

        bidText = new TextField("bidtext", "", 10, TextField.NUMERIC);
        bidText.setString("");

        bidForm = new Form("bidform");
        bidForm.setTitle("Someter una oferta para:");
        BID_PROPERTY = 0;
        bidForm.append(item);
        bidForm.append(new StringItem("", "Su oferta:"));
        bidForm.append(bidText);
        bidForm.addCommand(cancelCommand);
        bidForm.addCommand(goCommand);
        bidForm.setCommandListener(this);
        display.setCurrent(bidForm);
    }

    /**
     * Actualice la tarjeta de listas con la
     * lista actual de ofertas en que estamos interesados.
     */
    public void getNewBids()
    {
        // Restablezca la lista anterior.
        listings = null;
        listings = new List("Ofertas de JdbcMe", Choice.IMPLICIT);
        java.sql.Connection     conn = null;
        Statement               stmt = null;
                            try
        {
            conn = DriverManager.getConnection(urlText.getString() + ";" +
jdbcmeText.getString());

            stmt = conn.createStatement();

            // Dado que no deseamos que persista la sentencia preparada,
            // una sentencia normal es más adecuada en este entorno.
            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 se ha encontrado ninguna oferta", null);
            listings.addCommand(backCommand);
            listings.setCommandListener(this);
        }
    }
    catch (Exception e)
    {
        // Actualmente no se ha recuperado ninguna lista válida,
        // por lo que la restableceremos como vacía.
        listings = new List("Ofertas de JdbcMe", Choice.IMPLICIT);
        listings.append(GETBIDS, null);
        listings.addCommand(backCommand);
        listings.setCommandListener(this);

         // Vuelva al proceso main tras mostrar el error.
        showError(main, e);
        return;
    }
  finally
    {
        if (conn != null)
        {
                        try
            {
                conn.close();
            }
    catch (Exception e)
            {
            }
        }
        conn = null;
        stmt = null;
    }
    showBids();
}

public void setupListingsFromOfflineData()
{
    // Sáltese las 4 primeras filas del almacén de registros
    // (marca de formato, versión, número de columnas y tipos
    // de columna sql)
    // y cada fila posterior del almacén de registros es
    // una sola columna. La consulta devuelve 3 columnas
    // que devolveremos concatenadas como una sola serie.
    ResultSet         rs = null;
    listings.addCommand(backCommand);
    listings.setCommandListener(this);
                        try
    {
        int                     i = 5;
        int                     max = 0;
        StringBuffer   buf  = new StringBuffer(20);

        // Creador y tipo de base de datos no utilizados en MIDP
        rs = new JdbcMeOfflineResultSet("JdbcMidpBidListings", 0, 0);
        if (rs == null)
        {
            // Nuevas listas...
            listings = new List("Ofertas de JdbcMe", 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 se ha encontrado ninguna oferta", null);
        return;
        }
    }
    catch (Exception e)
    {
        // Actualmente no se ha recuperado ninguna lista válida,
        // por lo que la restableceremos como vacía.
        listings = new List("Ofertas de JdbcMe", Choice.IMPLICIT);
        listings.append(GETBIDS, null);
        listings.addCommand(backCommand);
        listings.setCommandListener(this);

        // Vuelva al proceso main tras mostrar el error.
        showError(main, e);
        return;
    }
  finally
    {
        if (rs != null)
        {
                        try
            {
                rs.close();
            }
    catch (Exception e)
            {
            }
            rs = null;
        }
        System.gc();
    }
}

/**
 * Actualice la tarjeta de listas con la
```

```
     * lista actual de ofertas en que estamos interesados.
     */
    public void submitBid()
    {
        java.sql.Connection     conn = null;
        Statement               stmt = null;
                            try
        {
            conn = DriverManager.getConnection(urlText.getString() + ";" +
jdbcmeText.getString());

            stmt = conn.createStatement();

            // Dado que no deseamos que persista la sentencia preparada,
            // una sentencia normal es más adecuada en este entorno.
            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)
            {
                // Oferta aceptada.
                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
                {
                    // Creador y tipo de base de datos no utilizados en 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();
                }

                // Actualice también la lista dinámica de ese conjunto de
resultados.
                listings.set(bidRow, newS, null);
                display.setCurrent(listings);
                conn.commit();
            }
          else
            {
                conn.rollback();
                throw new SQLException("Error al ofertar, otra persona se lo
impidió");
```

```
            }
        }
         catch (SQLException e)
        {
            // Vuelva al formulario de oferta tras mostrar el error.
            showError(bidForm, e);
            return;
        }
      finally
        {
            if (conn != null)
            {
                            try
                {
                    conn.close();
                }
        catch (Exception e)
                {
                }
            }
            conn = null;
            stmt = null;
        }

        // Salga sin excepción y, a continuación, muestre las ofertas
actuales.
        showBids();
    }

    /**
     * Visualice una condición de error.
     */
    public void showError(Displayable d, Exception e)
    {
        String s = e.toString();

        onErrorGoBackTo = d;
        errorForm = new Form("Error");
        errorForm.setTitle("Error SQL");
        errorForm.append(new StringItem("", s));
        errorForm.addCommand(backCommand);
        errorForm.setCommandListener(this);
        display.setCurrent(errorForm);
    }

    /**
     * Visualice las ofertas actuales.
     */
    public void showBids()
    {
        if (listings == null)
        {
            // Si no tenemos listas actuales,
            // vamos a configurarlas.
            listings = new List("Ofertas de JdbcMe", Choice.IMPLICIT);
            setupListingsFromOfflineData();
        }
        display.setCurrent(listings);
    }

    /**
     * Tiempo para hacer una pausa; libere el espacio que no sea necesario
```

```
en este momento.
     */
    public void pauseApp()
    {
        display.setCurrent(null);
    }

    /**
     * La operación de destrucción debe efectuar una limpieza completa.
     */
    public void destroyApp(boolean unconditional)
    {
    }
}«
```

Ejemplo: cómo se utiliza ToolboxME para iSeries, MIDP y Toolbox para Java

El fuente siguiente muestra un modo en que la aplicación de ToolboxME para iSeries puede utilizar MIDP (Mobile Information Device Profile) e IBM Toolbox para Java a fin de acceder a los datos y servicios de un servidor iSeries.

Este ejemplo constituye una demostración de cada una de las funciones incorporadas en el soporte de IBM Toolbox para Java 2 Micro Edition. Esta aplicación presenta varias páginas o pantallas que ilustran algunas de las muchas formas en que el dispositivo Tier0 puede utilizar estas funciones.

Una vez compilado como un programa de trabajo, el código de ejemplo siguiente utiliza un archivo PCML (Program Call Markup Language) para ejecutar mandatos en el servidor iSeries.

Para crear una versión de trabajo del código fuente y obtener el fuente PCML necesario para ejecutar mandatos del servidor, debe bajar el ejemplo. Puede que también desee consultar las instrucciones para crear y ejecutar el programa de ejemplo.

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de ToolboxME para iSeries. Este programa es un ejemplo que
muestra cómo
// ToolboxME para iSeries puede emplear PCML para acceder a los datos y
servicios
// de un servidor iSeries.
//
// Esta aplicación requiere que el archivo qsyrusri.pcml esté presente en la
// CLASSPATH de 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_;

    // Un objeto del sistema ToolboxME.
    private AS400 system_;

    private List      main_ = new List("Demostración de MIDP de la Caja de
herramientas", Choice.IMPLICIT);

    // Cree un formulario para cada componente.
    private Form      signonForm_;
```

```
    private Form      cmdcallForm_;
    private Form      pgmcallForm_;
    private Form      dataqueueForm_;
    private Form      aboutForm_;

    // Texto visualizable para cada componente.
    static final String SIGN_ON       = "Inicio de sesión";
    static final String COMMAND_CALL  = "Llamada a mandato";
    static final String PROGRAM_CALL  = "Llamada a programa";
    static final String DATA_QUEUE    = "Cola de datos";
    static final String ABOUT         = "Acerca de";

    static final String NOT_SIGNED_ON = "No se ha iniciado la sesión.";
    static final String DQ_READ       = "Leer";
    static final String DQ_WRITE      = "Escribir";

    // Un código para visualizar el estado de inicio de sesión.
    private Ticker    ticker_ = new Ticker(NOT_SIGNED_ON);

    // Mandatos que pueden ejecutarse.
    private static final Command actionExit_   = new Command("Salir",
Command.SCREEN, 0);
    private static final Command actionBack_   = new Command("Atrás",
Command.SCREEN, 0);
    private static final Command actionGo_     = new Command("Ir",
Command.SCREEN, 1);
    private static final Command actionClear_  = new Command("Borrar",
Command.SCREEN, 1);
    private static final Command actionRun_    = new Command("Ejecutar",
Command.SCREEN, 1);
    private static final Command actionSignon_ = new Command(SIGN_ON,
Command.SCREEN, 1);
    private static final Command actionSignoff_= new Command("Fin de
sesión", Command.SCREEN, 1);

    private Displayable   onErrorGoBackTo_;  // Formulario al que se volverá
al terminar de visualizar el formulario de error.

    // Campos TextField para el formulario de inicio de sesión.
    private TextField signonSystemText_ = new TextField("Sistema",
"rchasdm3", 20, TextField.ANY);
    private TextField signonUidText_ = new TextField("ID de usuario",
"JAVA", 10, TextField.ANY);
    private TextField signonPwdText_ = new TextField("Contraseña", "JTEAM1",
10, TextField.PASSWORD);  // TBD temporal
    private TextField signonServerText_ = new TextField("MEServer",
"localhost", 10, TextField.ANY);
    private StringItem signonStatusText_ = new StringItem("Estado",
NOT_SIGNED_ON);

    // Campos TextField para el formulario de llamada a mandato.
    private TextField cmdText_ = new TextField("Mandato", "CRTLIB FRED",
256, TextField.ANY);  // TBD: tamaño máximo; TBD: cuadro de texto???
    private StringItem cmdMsgText_ = new StringItem("Mensajes", null);
    private StringItem cmdStatusText_ = new StringItem("Estado", null);

    // Campos TextField para el formulario de llamada a programa.
    private StringItem pgmMsgDescription_ = new StringItem("Mensajes",
null);
    private StringItem pgmMsgText_ = new StringItem("Mensajes", null);

    // Campos TextField para el formulario de cola de datos.
```

```
    private TextField dqInputText_ = new TextField("Datos a escribir",
"Hola", 30, TextField.ANY);
    private StringItem dqOutputText_ = new StringItem("Contenido de cola de
datos", null);
    private ChoiceGroup dqReadOrWrite_ = new ChoiceGroup("Acción",
Choice.EXCLUSIVE, new String[] { DQ_WRITE, DQ_READ}, null);
    private StringItem dqStatusText_ = new StringItem("Estado", null);


    /**
     * Crea un nuevo ToolboxMidpDemo.
     **/
    public ToolboxMidpDemo()
    {
        display_ = Display.getDisplay(this);
        // Nota: la demostración KVM utilizaba TabbedPane para el panel
principal. MIDP no tiene ninguna clase parecida, por que en su lugar se
utiliza List.
    }

    /**
     * Visualice la pantalla principal.
     * Implementa el método abstracto de la clase 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_);
    }

    // Implementa el método de la interfaz CommandListener.
    public void commandAction(Command action, Displayable dsp)
    {
        // Todo el proceso de 'exit' y 'back' es el mismo.
        if (action == actionExit_)
        {
            destroyApp(false);

            notifyDestroyed();
        }
        else if (action == actionBack_)
        {
            // Vuelva al menú principal.
            display_.setCurrent(main_);
        }
        else if (dsp instanceof List)
        {
            List current = (List)dsp;

            // Se ha producido una acción en la página principal.
            if (current == main_)
            {
                int   idx = current.getSelectedIndex();
```

```
                switch (idx)
                {
                case 0:     // Inicio de sesión
                    showSignonForm();
                break;
                case 1:     // Llamada a mandato
                    showCmdForm();
                break;
                case 2:     // Llamada a programa
                    showPgmForm();
                break;
                case 3:     // Cola de datos
                    showDqForm();
                break;
                case 4:     // Acerca de
                    showAboutForm();
                break;
                default:    // Ninguna de las opciones anteriores
                    feedback("Error interno: índice seleccionado sin manejar
en principal: " + idx, AlertType.ERROR);
                break;
                }
            } // current == main
          else
                feedback("Error interno: el objeto visualizable es una lista
pero no es main_.", AlertType.ERROR);
        } // instanceof List
        else if (dsp instanceof Form)
        {
            Form current = (Form)dsp;

            if (current == signonForm_)
            {
                if (action == actionSignon_)
                {
                    // Cree un objeto de sistema ToolboxME.
                    system_ = new AS400(signonSystemText_.getString(),
signonUidText_.getString(), signonPwdText_.getString(),
signonServerText_.getString());

                            try
                    {
                        // Conéctese al iSeries.
                        system_.connect();

                        // Establezca el texto de estado de inicio de
sesión.
                        signonStatusText_.setText("Se ha iniciado la
sesión.");

                        // Visualice un diálogo de confirmación de que el
usuario tiene iniciada la sesión.
                        feedback("Inicio de sesión satisfactorio.",
AlertType.INFO, main_);

                        // Sustituya el botón de inicio de sesión por el de
fin de sesión.
                        signonForm_.removeCommand(actionSignon_);
                        signonForm_.addCommand(actionSignoff_);

                        // Actualice el código.
                        ticker_.setString("... Sesión iniciada en '" +
```

```
signonSystemText_.getString() + "' como '" + signonUidText_.getString() + "'
mediante '" + signonServerText_.getString() + "' ... ");
                    }
        catch (Exception e)
                    {
      e.printStackTrace();

                        // Establezca el texto de estado de inicio de
sesión.
                        signonStatusText_.setText(NOT_SIGNED_ON);

                        feedback("Inicio de sesión anómalo.  " +
e.getMessage(), AlertType.ERROR);
                    }
                }
                else if (action == actionSignoff_)
                {
      if (system_ == null)
                        feedback("Error interno: el sistema es nulo.",
AlertType.ERROR);
          else
                    {
                        try
                        {
                            // Desconéctese del iSeries.
                            system_.disconnect();
                            system_ = null;

                            // Establezca el texto de estado de inicio de
sesión.
                            signonStatusText_.setText(NOT_SIGNED_ON);

                            // Visualice un diálogo de confirmación de que
el usuario ya no tiene iniciada la sesión.
                            feedback("Fin de sesión satisfactorio.",
AlertType.INFO, main_);

                            // Sustituya el botón de fin de sesión por el de
inicio de sesión.
                            signonForm_.removeCommand(actionSignoff_);
                            signonForm_.addCommand(actionSignon_);

                            // Actualice el código.
                            ticker_.setString(NOT_SIGNED_ON);
                        }
        catch (Exception e)
                        {
                            feedback(e.toString(), AlertType.ERROR);

      e.printStackTrace();

                            signonStatusText_.setText("Error.");

                            feedback("Error al finalizar la sesión.",
AlertType.ERROR);
                        }
                    }
                }
                else  // Ninguna de las opciones anteriores.
                {
                    feedback("Error interno: no se reconoce la acción.",
AlertType.INFO);
```

```
            }
        } // signonForm_
        else if (current == cmdcallForm_)
        {
            if (action == actionRun_)
            {
                // Si el usuario no ha iniciado la sesión, visualice una
alerta.
      if (system_ == null)
                {
                    feedback(NOT_SIGNED_ON, AlertType.ERROR);
        return;
                }

                // Obtenga el mandato entrado por el usuario en el
dispositivo inalámbrico.
                String cmdString = cmdText_.getString();

                // Si no se ha especificado el mandato, visualice una
alerta.
                if (cmdString == null || cmdString.length() == 0)
                    feedback("Especificar mandato.", AlertType.ERROR);
          else
                {
                        try
                    {
         // Ejecute el mandato.
                        String[] messages = CommandCall.run(system_,
cmdString);

                        StringBuffer status = new StringBuffer("Mandato
completado con ");

                        // Compruebe si hay algún mensaje.
                        if (messages.length == 0)
                        {
                            status.append("0 mensajes devueltos.");

                            cmdMsgText_.setText(null);

                            cmdStatusText_.setText("Mandato completado
satisfactoriamente.");
                        }
          else
                        {
                            if (messages.length == 1)
                                status.append("1 mensaje devuelto.");
          else
                                status.append(messages.length + "
mensajes devueltos.");

                            // Si hay mensajes, visualice sólo el primer
mensaje.
                            cmdMsgText_.setText(messages[0]);

                            cmdStatusText_.setText(status.toString());
                        }

                        repaint();
                    }
        catch (Exception e)
                    {
```

```
                            feedback(e.toString(), AlertType.ERROR);

      e.printStackTrace();

                            feedback("Error al ejecutar el mandato.",
AlertType.ERROR);
                        }
                    }
                }
                else if (action == actionClear_)
                {
                    // Borre el texto y los mensajes del mandato.
                    cmdText_.setString("");

                    cmdMsgText_.setText(null);

                    cmdStatusText_.setText(null);

                    repaint();
                }
                else  // Ninguna de las opciones anteriores.
                {
                    feedback("Error interno: no se reconoce la acción.",
AlertType.INFO);
                }
            } // cmdcallForm_
            else if (current == pgmcallForm_)
            {
                if (action == actionRun_)
                {
                    // Si el usuario no tiene iniciada la sesión antes de
hacer una llamada a programa, visualice una alerta.
      if (system_ == null)
                    {
                        feedback(NOT_SIGNED_ON, AlertType.ERROR);
            return;
                    }

                    pgmMsgText_.setText(null);

                    // Consulte el ejemplo de PCML en la guía del
programador de la Caja de Herramientas.
                    String pcmlName = "qsyrusri.pcml"; // El archivo PCML
que se desea utilizar.
                    String apiName = "qsyrusri";

                    // Cree una tabla hash que contenga los parámetros de
entrada de la llamada a programa.
                    Hashtable parmsToSet = new Hashtable(2);
                    parmsToSet.put("qsyrusri.receiverLength", "2048");
                    parmsToSet.put("qsyrusri.profileName",
signonUidText_.getString().toUpperCase());

                    // Cree una matriz de tipo serie que contenga los
parámetros de salida que deben recuperarse.
                    String[] parmsToGet = { "qsyrusri.receiver.userProfile",
                        "qsyrusri.receiver.previousSignonDate",
                        "qsyrusri.receiver.previousSignonTime",
                        "qsyrusri.receiver.daysUntilPasswordExpires"};

                    // Una matriz de tipo serie con las descripciones de los
parámetros que deben visualizarse.
```

```
                    String[] displayParm = { "Perfil", "Fecha de último
inicio de sesión", "Hora de último inicio de sesión", "Caducidad de
contraseña (días)"};

                            try
                    {
         // Ejecute el programa.
                        String[] valuesToGet = ProgramCall.run(system_,
pcmlName, apiName, parmsToSet, parmsToGet);

                        // Cree un StringBuffer y añada cada uno de los
parámetros recuperados.
                        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");

                        // Establezca el texto visualizable de los
resultados de la llamada a programa.
                        pgmMsgText_.setText(txt.toString());

                        StringBuffer status = new StringBuffer("Programa
completado con ");

                        if (valuesToGet.length == 0)
                        {
                            status.append("0 valores devueltos.");

                            feedback(status.toString(), AlertType.INFO);
                        }
          else
                        {
                            if (valuesToGet.length == 1)
                                status.append("1 valor devuelto.");
          else
                                status.append(valuesToGet.length + " valores
devueltos.");

                            feedback(status.toString(), AlertType.INFO);
                        }
                    }
        catch (Exception e)
                    {
                        feedback(e.toString(), AlertType.ERROR);

      e.printStackTrace();

                        feedback("Error al ejecutar el programa.",
AlertType.ERROR);
                    }
                }
                else if (action == actionClear_)
                {
```

```
                    // Borre los resultados de la llamada a programa.
                    pgmMsgText_.setText(null);

                    repaint();
                }
            }  // pgmcallForm_
            else if (current == dataqueueForm_)  // Cola de datos
            {
                if (action == actionGo_)
                {
                    // Si el usuario no ha iniciado la sesión antes de
efectuar acciones de cola de datos, visualice una alerta.
      if (system_ == null)
                    {
                        feedback(NOT_SIGNED_ON, AlertType.ERROR);

            return;
                    }

                    // Cree una biblioteca para crear la cola de datos en su
interior.
                            try
                    {
                        CommandCall.run(system_, "CRTLIB FRED");
                    }
        catch (Exception e)
                    {
                    }

                    // Ejecute un mandato para crear una cola de datos.
                            try
                    {
                        CommandCall.run(system_, "CRTDTAQ FRED/MYDTAQ
MAXLEN(2000)");
                    }
        catch (Exception e)
                    {
                        feedback("Error al crear la cola de datos.  " +
e.getMessage(), AlertType.WARNING);
                    }

                            try
                    {
                        // Consulte qué acción se ha seleccionado (Leer o
Escribir).
                        if
(dqReadOrWrite_.getString(dqReadOrWrite_.getSelectedIndex()).equals(DQ_WRITE))
                        {
                            // Escribir
                            dqOutputText_.setText(null);

                            // Obtenga el texto de la entrada del
dispositivo inalámbrico que debe escribirse en la cola de datos.
                            if (dqInputText_.getString().length() == 0)
                                dqStatusText_.setText("No se ha especificado
ningún dato.");
          else
                            {
                                // Escriba en la cola de datos.
                                DataQueue.write(system_,
"/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ", dqInputText_.getString().getBytes() );
```

```
                                dqInputText_.setString(null);

                                // Visualice el estado.
                                dqStatusText_.setText("Operación de
escritura completada.");
                            }
                        }
                        else  // Leer
                        {
                            // Lea información de la cola de datos.
                            byte[] b = DataQueue.readBytes(system_,
"/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");

                            // Determine si la cola de datos contenía o no
entradas y visualice el mensaje adecuado.
                            if (b == null)
                            {
                                dqStatusText_.setText("No hay ninguna
entrada de cola de datos disponible.");

                                dqOutputText_.setText(null);
                            }
                            else if (b.length == 0)
                            {
                                dqStatusText_.setText("La entrada de cola de
datos no tiene datos.");

                                dqOutputText_.setText(null);
                            }
          else
                            {
                                dqStatusText_.setText("Operación de lectura
completada.");

                                dqOutputText_.setText(new String(b));
                            }
                        }

                        repaint();
                    }
        catch (Exception e)
                    {
      e.printStackTrace();

                        feedback(e.toString(), AlertType.ERROR);

                        feedback("Error al ejecutar el mandato.  " +
e.getMessage(), AlertType.ERROR);
                    }
                }  // actionGo_
                else if (action == actionClear_)
                {
                    // Borre el formulario de cola de datos.
                    dqInputText_.setString("");

                    dqOutputText_.setText(null);

                    dqReadOrWrite_.setSelectedFlags(new boolean[] { true,
false});

                    dqStatusText_.setText(null);
```

```
                    repaint();
                }
                else  // Ninguna de las opciones anteriores.
                {
                    feedback("Error interno: no se reconoce la acción.",
AlertType.INFO);
                }
            } // dataqueueForm_
            else if (current == aboutForm_)  // "Acerca de".
            {
                // Nunca debería llegar aquí, ya que el único botón es
"Atrás".
            }  // Ninguna de las opciones anteriores.
          else
                feedback("Error interno: no se reconoce el formulario.",
AlertType.ERROR);
        } // instanceof Form
          else
            feedback("Error interno: no se reconoce el objeto
visualizable.", AlertType.ERROR);
    }



    /**
     *  Visualiza el formulario "Acerca de".
     **/
    private void showAboutForm()
    {
        // Si el formulario de acerca de es nulo, créelo y añádalo.
        if (aboutForm_ == null)
        {
            aboutForm_ = new Form(ABOUT);
            aboutForm_.append(new StringItem(null, "Esta es una aplicación
de ejemplo de MIDP que utiliza Toolbox Micro Edition (ToolboxME)."));

            aboutForm_.addCommand(actionBack_);
            aboutForm_.setCommandListener(this);
        }

        display_.setCurrent(aboutForm_);
    }



    /**
     *  Visualiza el formulario "Inicio de sesión".
     **/
    private void showSignonForm()
    {
        // Cree el formulario de inicio de sesión.
        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_);
    }



    /**
     *  Visualiza el formulario "Llamada a mandato".
     **/
    private void showCmdForm()
    {
        // Cree el formulario de llamada a mandato.
        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_);
    }



    /**
     *  Visualiza el formulario "Llamada a programa".
     **/
    private void showPgmForm()
    {
        // Cree el formulario de llamada a programa.
        if (pgmcallForm_ == null)
        {
            pgmcallForm_ = new Form(PROGRAM_CALL);
            pgmcallForm_.append(new StringItem(null, "Esto llama a la API
Recuperar información de usuario (QSYRUSRI) y devuelve información sobre el
perfil de usuario actual."));
            pgmcallForm_.append(pgmMsgText_);
            pgmcallForm_.addCommand(actionBack_);
            pgmcallForm_.addCommand(actionClear_);
            pgmcallForm_.addCommand(actionRun_);
            pgmcallForm_.setCommandListener(this);
            pgmcallForm_.setTicker(ticker_);
        }

        display_.setCurrent(pgmcallForm_);
    }


    /**
     *  Visualiza el formulario "Cola de datos".
     **/
    private void showDqForm()
    {
        // Cree el formulario de cola de datos.
```

```
        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());
    }

    /**
     *  Este método se utiliza para crear un diálogo y mostrar información
al usuario mediante una alerta.
     **/
    private void feedback(String text, AlertType type, Displayable
returnToForm)
    {
        System.err.flush();
        System.out.flush();

        Alert alert = new Alert("Alerta", text, null, type);

        if (type == AlertType.INFO)
            alert.setTimeout(3000);  // milisegundos
          else
            alert.setTimeout(Alert.FOREVER);  // Solicite al usuario que
desestime la alerta.

        display_.setCurrent(alert, returnToForm);
    }


    // Fuerce a que se vuelva a generar el formulario actual.
    private void repaint()
    {
        Alert alert = new Alert("Actualizando visualización...", null, null,
AlertType.INFO);
        alert.setTimeout(1000);  // milisegundos

        display_.setCurrent(alert, display_.getCurrent());
    }



    /**
     * Tiempo para hacer una pausa; libere el espacio que no sea necesario
en este momento.
     * Implementa el método abstracto de la clase Midlet.
     **/
```

```
    protected void pauseApp()
    {
        display_.setCurrent(null);
    }



    /**
     * La operación de destrucción debe efectuar una limpieza completa.
     * Implementa el método abstracto de la clase Midlet.
     **/
    protected void destroyApp(boolean unconditional)
    {
        // Desconéctese del iSeries si se destruye o abandona Midlet.
        if (system_ != null)
        {
                            try
            {
                system_.disconnect();
            }
        catch (Exception e)
            {
            }
        }
    }
}
```

Ejemplos: clases de utilidades

En esta sección figura una lista de los ejemplos de código que se proporcionan en toda la documentación de las clases de utilidades.

IBM Toolbox Installer

- Ejemplo: cómo se utiliza la clase AS400ToolboxInstaller
- Ejemplo: instalar IBM Toolbox para Java con la clase AS400ToolboxInstaller
- Ejemplo: instalar el paquete ACCESS desde la línea de mandatos
- Ejemplo: trabajar con la clase Caja de Herramientas Gráfica desde la línea de mandatos

JarMaker

- Ejemplo: extraer AS400.class, y todas sus clases dependientes, de jt400.jar
- Ejemplo: subdividir jt400.jar en un conjunto de archivos de 300 K
- Ejemplo: eliminar archivos no utilizados de un archivo JAR
- Ejemplo: crear un archivo JAR que tenga 400 Kbytes menos a base de omitir las tablas de conversión con el parámetro -ccsid

»CommandPrompter

- Ejemplo: cómo se utiliza CommandPrompter para solicitar y ejecutar un mandato«

La siguiente declaración de limitación de responsabilidad es válida para todos los ejemplos de IBM Toolbox para Java:

Ejemplo: cómo se utiliza AS400ToolboxInstaller para instalar y actualizar IBM Toolbox para Java

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de instalación/actualización. Este programa utiliza la clase
AS400ToolboxInstaller
// para instalar y actualizar el paquete de IBM Toolbox para Java en la
estación de trabajo.
//
// El programa comprueba la vía de acceso destino del paquete de IBM Toolbox
para Java.
// Si no se encuentra el paquete, instala el paquete en la estación de
trabajo.
// Si se encuentra, examina si hay actualizaciones en la vía de acceso
origen.
// Si se encuentran actualizaciones, se copian en la estación de trabajo.
//
// Sintaxis de mandato:
//    checkToolbox origen destino
//
// Donde
//    origen = ubicación de los archivos fuente. Este nombre tiene formato
de URL.
//    destino = ubicación de los archivos destino.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import utilities.*;

public class checkToolbox extends Object
{
  public static void main(String[] parameters)
   {
      System.out.println( " " );



      // Prosiga con la instalación/actualización únicamente si se han
      // especificado tanto el nombre de origen como el nombre de destino.

        if (parameters.length >= 2)
      {

         // El primer parámetro es el origen de los archivos; el segundo es
el destino.

         String sourcePath = parameters[0];
         String targetPath = parameters[1];

         boolean installIt = false;
         boolean updateIt  = false;
```

```
         // Se ha creado un lector para leer la entrada del usuario.

      BufferedReader inputStream = new BufferedReader(new
InputStreamReader(System.in),1);

                              try
         {
            // Apunte al paquete de origen. AS400ToolboxInstaller utiliza
            // la clase URL para acceder a los archivos.

            URL sourceURL  = new URL(sourcePath);



            // Vea si el paquete está instalado en el cliente. Si no,
solicite
            // al usuario si debe realizarse la instalación en este momento.

            if (AS400ToolboxInstaller.isInstalled("ACCESS", targetPath) ==
false)
            {
               System.out.print("IBM Toolbox para Java no está instalado.
Instalar ahora (Y/N):");

                  String userInput = inputStream.readLine();

                  if ((userInput.charAt(0) == 'y')   ||
                      (userInput.charAt(0) == 'Y'))
                  installIt = true;
            }



            // El paquete está instalado. Vea si es necesario copiar
actualizaciones desde
            // el servidor. Si el destino no está actualizado, solicite al
usuario si debe
            // realizarse la actualización en este momento.

          else
            {
               if (AS400ToolboxInstaller.isUpdateNeeded("ACCESS",
targetPath, sourceURL) == true)
               {
                  System.out.print("IBM Toolbox para Java no está
actualizado. Instalar arreglos (Y/N):");

                  String userInput = inputStream.readLine();

                  if ((userInput.charAt(0) == 'y')   ||
                      (userInput.charAt(0) == 'Y'))
                     updateIt = true;
               }
          else
                  System.out.println("El directorio destino está
actualizado, no es necesaria ninguna actualización.");
            }



            // Si es necesario instalar o actualizar el paquete,
```

```
            if (updateIt || installIt)
            {
               // copie los archivos del servidor en el destino.

               AS400ToolboxInstaller.install("ACCESS", targetPath,
sourceURL);




               // Informe de la correcta finalización de la
instalación/actualización.

               System.out.println(" ");

               if (installIt)
                  System.out.println("Instalación correcta.");
          else
                  System.out.println("Actualización correcta.");



               // Indique al usuario qué debe añadirse a la variable de
entorno
               // CLASSPATH.

               Vector classpathAdditions =
AS400ToolboxInstaller.getClasspathAdditions();

               if (classpathAdditions.size() > 0)
               {
System.out.println("");                  System.out.println("Añada lo
siguiente a la variable de entorno CLASSPATH:");

                  for (int i = 0; i < classpathAdditions.size(); i++)
                  {
System.out.print ("  ");
System.out.println((String)classpathAdditions.elementAt(i));
                  }
               }



               // Indique al usuario qué puede eliminarse de la variable de
entorno
               // CLASSPATH.

               Vector classpathRemovals =
AS400ToolboxInstaller.getClasspathRemovals();

               if (classpathRemovals.size() > 0)
               {
System.out.println("");                  System.out.println("Elimine lo
siguiente de la variable de entorno CLASSPATH:");

                  for (int i = 0; i < classpathRemovals.size(); i++)
                  {
System.out.print ("  ");
System.out.println((String)classpathRemovals.elementAt(i));
                  }
               }
            }
```

```
         }

        catch (Exception e)
         {
            // Si alguna de las operaciones anteriores ha fallado, indique
que la
            // operación ha fallado y envíe a la salida la excepción.

            System.out.println("La instalación/actualización ha fallado");
        System.out.println(e);
         }
      }



        // Visualice texto de ayuda si los parámetros son incorrectos.

          else
       {
System.out.println("");System.out.println("");System.out.println("");
System.out.println("Los parámetros no son correctos. La sintaxis del mandato
es:");
System.out.println("");         System.out.println("  checkToolbox víaOrigen
víaDestino");
System.out.println("");         System.out.println("Donde");
System.out.println("");         System.out.println("   víaOrigen = origen de
los archivos de IBM Toolbox para Java");
         System.out.println("   víaDestino = destino de los archivos de IBM
Toolbox para Java");
System.out.println("");            System.out.println("Por ejemplo:");
System.out.println("");         System.out.println("   checkToolbox
http://mySystem/QIBM/ProdData/HTTP/Public/jt400/ d:\\jt400");
System.out.println("");System.out.println("");      }

       System.exit(0);
   }
}
```

»

Ejemplo: cómo se utiliza CommandPrompter

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////////
//
// Ejemplo de CommandPrompter. Este programa utiliza CommandPrompter,
CommandCall y
// AS400Message para solicitar un mandato, ejecutar el mandato y visualizar
los
// mensajes devueltos si el mandato no se ejecuta.
//
// Sintaxis de mandato:
//    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];

  // Inicie CommandPrompter
  CommandPrompter cp = new CommandPrompter(frame, system, cmdName);
  if (cp.showDialog() == CommandPrompter.OK)
    {
    String cmdString = cp.getCommandString();
    System.out.println("Serie del mandato: " + cmdString);

    // Ejecute el mandato generado en el programa de solicitud.
    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);
  }
}
```

«

Ejemplos: clases de vaccess

En esta sección figura una lista de los ejemplos de código que se proporcionan en toda la documentación de las clases de vaccess.

AS400Panes

- Ejemplo: crear una sección AS400DetailsPane para presentar la lista de usuarios definidos en systemAS400DetailsPane
- Ejemplo: cargar el contenido de una sección de detalles antes de añadirla a un marco
- Ejemplo: cómo se utiliza una sección AS400ListPane para presentar una lista de usuarios
- Ejemplo: cómo se utiliza una sección AS400DetailsPane para visualizar los mensajes devueltos en una llamada a mandato
- Ejemplo: cómo se utiliza una sección AS400TreePane para visualizar una vista en árbol de un directorio
- Ejemplo: cómo se utiliza una sección AS400ExplorerPane para presentar diversos recursos de impresión

Llamada a mandato

- Ejemplo: crear un botón CommandCallButton
- Ejemplo: añadir el escuchador ActionCompletedListener para procesar todos los mensajes de iSeries generados por un mandato
- Ejemplo: cómo se utiliza el elemento de menú CommandCallMenuItem

Colas de datos

- Ejemplo: crear un DataQueueDocument
- Ejemplo: cómo se utiliza un DataQueueDocument

Eventos de error

- Ejemplo: manejar eventos de error
- Ejemplo: definir un escuchador de errores
- Ejemplo: cómo se utiliza un manejador personalizado para manejar eventos de error

JDBC

- Ejemplo: cómo se utiliza el controlador JDBC para crear y llenar con datos una tabla
- Ejemplo: cómo se utiliza el controlador JDBC para consultar una tabla y enviar su contenido a la salida
- Ejemplo: crear un objeto AS400JDBCDataSourcePane

Trabajos

- Ejemplo: crear una lista VJobList y presentarla en una sección AS400ExplorerPane
- Ejemplo: presentar una lista de trabajos en una sección de explorador

Llamada a programa

- Ejemplo: crear un ProgramCallMenuItem
- Ejemplo: procesar todos los mensajes de iSeries generados por programa
- Ejemplo: añadir dos parámetros
- Ejemplo: cómo se utiliza un botón ProgramCallButton en una aplicación

Acceso a nivel de registro

- Ejemplo: crear un objeto RecordListTablePane para visualizar todos los registros menores o iguales que una clave

SpooledFileViewer

- Ejemplo: crear un visor de archivos en spool para ver un archivo en spool previamente creado en un iSeries

Valores del sistema

- Ejemplo: crear una GUI de valores del sistema utilizando una sección AS400ExplorerPane

Usuarios y grupos

- Ejemplo: crear una lista VUserList con la sección AS400DetailsPane
- Ejemplo: cómo se utiliza una sección AS400ListPane para crear una lista de usuarios que permita realizar una selección

Ejemplo: cómo se utiliza VUserList

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de VUserList. Este programa presenta una lista de usuarios de
// un sistema en una sección de lista y permite la selección de uno o más
// usuarios.
//
// Sintaxis de mandato:
//    VUserListExample sistema
//
///////////////////////////////////////////////////////////////////////

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)
    {
        // De no haberse especificado un sistema, visualizar texto de ayuda
y
        // salir.
        if (args.length != 1)
        {
            System.out.println("Utilización:  VUserListExample sistema");
            return;
        }

                          try
        {
            // Cree un objeto AS400. El nombre del sistema se pasa
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // Cree un objeto VUserList. Representa una lista de usuarios
            // visualizados en la sección de lista.
     VUserList userList = new VUserList (system);

            // Cree un marco.
            JFrame f = new JFrame ("Ejemplo de VUserList");

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Cree una sección de lista para visualizar la lista de
usuarios.
```

```
            // Utilice load para obtener la información del servidor.
            listPane = new AS400ListPane (userList);
            listPane.addErrorListener (errorHandler);
     listPane.load ();

            // Cuando el marco se cierre, informe de los usuarios
            // seleccionados y salga.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    reportSelectedUsers ();
                    System.exit (0);
                }
            });

            // Diseñe el marco con la sección de lista.
        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 se ha seleccionado ningún usuario.");
          else
        {
            System.out.println ("Los usuarios seleccionados son:");
            for (int i = 0; i < selectedUsers.length; ++i)
                System.out.println (selectedUsers[i]);
        }
    }


}
```

Ejemplo: cómo se utiliza VMessageList

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de VMessageList. Este programa presenta una vista de detalles
// de los mensajes devueltos por una llamada a mandato.
//
// Sintaxis de mandato:
//    VMessageListExample sistema
//
// Este fuente es un ejemplo de "VMessageList" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////

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)
    {
        // De no haberse especificado un sistema, visualizar texto de ayuda
y
        // salir.
        if (args.length != 1)
        {
            System.out.println("Utilización:  VMessageListExample sistema");
            return;
        }

                          try
        {
            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // Cree un objeto CommandCall y ejecute el mandato.
            CommandCall command = new CommandCall (system);
            command.run ("CRTLIB FRED");

            // Cree un objeto VMessageList con los mensajes
            // devueltos por la llamada a mandato.
            VMessageList messageList = new VMessageList
(command.getMessageList ());

            // Cree un marco.
            JFrame f = new JFrame ("Ejemplo de VMessageList");

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);
```

```
            // Cree una sección de detalles para visualizar la lista de
mensajes.
            // Utilice load para cargar la información.
            AS400DetailsPane detailsPane = new AS400DetailsPane
(messageList);
            detailsPane.addErrorListener (errorHandler);
            detailsPane.load ();

            // Cuando el marco se cierre, salga.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Diseñe el marco con la sección de panel de detalles.
        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);
        }
    }


}
```

Ejemplo: cómo se utiliza VIFSDirectory

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de VIFSDirectory. Este programa presenta una vista de árbol de
// algunos directorios en el sistema de archivos integrado.
//
// Sintaxis de mandato:
//    VIFSDirectoryExample sistema
//
// Este fuente es un ejemplo de "VIFSDirectory" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////

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)
    {
        // De no haberse especificado un sistema, visualizar texto de ayuda
y
        // salir.
        if (args.length != 1)
        {
            System.out.println("Utilización:  VIFSDirectoryExample
sistema");
            return;
        }

                        try
        {
            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // Cree un objeto VIFSDirectory que representa la raíz
            // del árbol de directorios que vamos a mostrar.
            VIFSDirectory directory = new VIFSDirectory (system,
"/QIBM/ProdData");

            // Cree un marco.
            JFrame f = new JFrame ("Ejemplo de VIFSDirectory");

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Cree una sección en árbol para mostrar los directorios
jerárquicamente.
```

```
            // Cargue la información del sistema.
            AS400TreePane treePane = new AS400TreePane (directory);
            treePane.addErrorListener (errorHandler);
     treePane.load ();

            // Cuando el marco se cierre, salga.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Diseñe el marco con la sección en árbol.
        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);
        }
    }


}
```

Ejemplo: cómo se utiliza VPrinters

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de VPrinters. Este programa presenta diversos
// recursos de impresión de red con una sección de panel explorador.
//
// Sintaxis de mandato:
//    VPrintersExample sistema
//
// Este fuente es un ejemplo de "VPrinters" de IBM Toolbox para Java.
//
///////////////////////////////////////////////////////////////////////

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)
    {
        // De no haberse especificado un sistema, visualizar texto de ayuda
y
        // salir.
        if (args.length != 1)
        {
            System.out.println("Utilización:  VPrintersExample sistema");
            return;
        }

                          try
        {
            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // Cree un objeto VPrinters que representa la lista
            // de impresoras conectadas al sistema.
            VPrinters printers = new VPrinters (system);


            // Cree un marco.
            JFrame f = new JFrame ("Ejemplo de VPrinters");

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Cree una sección de panel explorador que presente los
recursos de impresión de red.
            // Utilice load para cargar la información desde el sistema.
```

```
            AS400ExplorerPane explorerPane = new AS400ExplorerPane
(printers);
            explorerPane.addErrorListener (errorHandler);
                    explorerPane.load ();

            // Cuando el marco se cierre, salga.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Diseñe el marco con la sección de panel explorador.
        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);
        }
    }


}
```

Ejemplo: cómo se utiliza el elemento de menú CommandCallMenuItem

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de elemento de menú de llamada a mandato. Este programa muestra
// cómo utilizar un elemento de menú que llama a un mandato del servidor.
// Visualizará los mensajes devueltos en un diálogo.
//
// Sintaxis de mandato:
//    CommandCallMenuItemExample sistema
//
///////////////////////////////////////////////////////////////////////

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)
    {
        // De no haberse especificado un sistema, visualizar texto de ayuda
y
        // salir.
        if (args.length != 1)
        {
            System.out.println("Utilización: CommandCallMenuItemExample
sistema");
            return;
        }

                          try
        {
            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // Cree un marco.
            f = new JFrame ("Ejemplo de elemento de menú de llamada a
mandato"

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Cree un objeto CommandCallMenuItem para ejecutar el mandato.
            CommandCallMenuItem menuItem = new CommandCallMenuItem ("Borrar
```

```
biblioteca FRED",
                null, system, "CLRLIB FRED"
             menuItem.addErrorListener (errorHandler);

             // Añada un escuchador de acciones completadas para visualizar
             // los mensajes devueltos en un diálogo.
     menuItem.addActionCompletedListener (new ActionCompletedListener ()
             {
           public void actionCompleted (ActionCompletedEvent event)
                {
                    // Obtenga la lista de mensajes del origen de evento.
                    CommandCallMenuItem item = (CommandCallMenuItem)
event.getSource ();
                    AS400Message[] messageList = item.getMessageList ();

                    // Utilice un objeto AS400DetailsPane para visualizar
los mensajes.
                    VMessageList vmessageList = new VMessageList
(messageList);
                    AS400DetailsPane messageDetails = new AS400DetailsPane
(vmessageList);
                    messageDetails.load ();

                    // Visualice los detalles en un diálogo.
                    JDialog dialog = new JDialog(f);
              dialog.getContentPane().setLayout(new BorderLayout());
              dialog.getContentPane().add("Center"messageDetails);
              dialog.pack();
              dialog.setVisible(true);
                }
             });

             // Cree un menú con el elemento.
             JMenu menu = new JMenu ("Llamadas a mandatos del servidor");
     menu.add (menuItem);

             JMenuBar menuBar = new JMenuBar ();
             menuBar.add (menu);

             f.getRootPane ().setJMenuBar (menuBar);

             // Cuando el marco se cierre, salga.
             f.addWindowListener (new WindowAdapter () {
                 public void windowClosing (WindowEvent event)
                 {
                     System.exit (0);
                 }
             });

             // Diseñe el marco con la sección de panel de detalles.
          f.getContentPane ().setLayout (new BorderLayout ());
             f.setSize (300, 400);
             f.show ();
          }
          catch (Exception e)
          {
             System.out.println ("Error: " + e.getMessage ());
             System.exit (0);
          }
       }
```

```
}
```

Ejemplo: cómo se utiliza DataQueueDocument

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
////////////////////////////////////////////////////////////////////////
//
// Ejemplo de documento de cola de datos. Este programa muestra cómo
// utilizar un documento asociado a una cola de datos del servidor.
//
// Sintaxis de mandato:
//    DataQueueDocumentExample sistema 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)
    {
        // De no haberse especificado un sistema o read|write, visualizar
        // texto de ayuda y salir.
        if (args.length != 2)
        {
            System.out.println("Utilización:  DataQueueDocumentExample
sistema read|write");
            return;
        }

        rw = args[1].equalsIgnoreCase ("read");
        String mode = rw ? "Read" : "Write";

                            try
        {
            // Cree dos marcos.
            JFrame f = new JFrame ("Ejemplo de documento de cola de datos -
" + mode);

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Cree un adaptador de cursor de trabajo. Esto ajustará el
            // cursor cada vez que se lea una cola de datos o se escriba en
ella.
            WorkingCursorAdapter cursorAdapter = new WorkingCursorAdapter
(f);

            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
```

```
        AS400 system = new AS400 (args[0]);

        // Cree el nombre de vía de acceso de la cola de datos.
        QSYSObjectPathName dqName = new QSYSObjectPathName ("QGPL",
            "JAVATALK", "DTAQ");

        // Compruebe que la cola de datos exista.
        DataQueue dq = new DataQueue (system, dqName.getPath ());
                        try
        {
            dq.create (200);
        }
      catch (Exception e)
        {
            // Omita las excepciones. Probablemente la cola de datos
            // ya exista.
        }

        // Cree un objeto DataQueueDocument.
        dqDocument = new DataQueueDocument (system, dqName.getPath ());
        dqDocument.addErrorListener (errorHandler);
        dqDocument.addWorkingListener (cursorAdapter);

        // Cree un campo de texto para presentar el documento.
        text = new JTextField (dqDocument, "", 40);
        text.setEditable (! rw);

        // Cuando se ejecute el programa, se necesitará controlar cuándo
        // se producen lecturas y escrituras. Para ello se utilizará
        // un botón.
        Button button = new Button (mode);
        button.addActionListener (new ActionListener ()
        {
            public void actionPerformed (ActionEvent event)
            {
                if (rw)
                    dqDocument.read ();
    else {
                    dqDocument.write ();
                    text.setText ("");
                }
            }
        });

        // Cuando el marco se cierre, salga.
        f.addWindowListener (new WindowAdapter () {
            public void windowClosing (WindowEvent event)
            {
                System.exit (0);
            }
        });

        // Diseñe el marco.
        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);
        }
    }
}
```

AS400JDBCDataSourcePane

La clase AS400JDBCDataSourcePane presenta los valores de propiedad de un objeto AS400JDBCDataSource. Si se desea, pueden efectuarse cambios en el objeto AS400JDBCDataSource.

AS400JDBCDataSourcePane amplía JComponent. Para utilizar un objeto AS400JDBCDataSourcePane a fin de visualizar las propiedades de un origen de datos, el origen de datos puede especificarse en el constructor de AS400JDBCDataSourcePane o establecerse después de que se haya creado AS400JDBCDataSourcePane mediante setDataSource(). Los cambios efectuados en la interfaz gráfica de usuario (GUI) pueden aplicarse al objeto de origen de datos con applyChanges().

En el ejemplo siguiente se crea un objeto AS400JDBCDataSourcePane y un botón **Aceptar** y se añaden a un marco. Los cambios realizados en la interfaz GUI se aplican al origen de datos al pulsar **Aceptar**.

»Ejemplo: cómo se utiliza AS400JDBCDataSourcePane

```
     // Cree un origen de datos.
         myDataSource = new AS400JDBCDataSource();

     // Cree una ventana para contener la sección y un botón Aceptar.
     JFrame frame = new JFrame ("Propiedades de origen de datos JDBC");

     // Cree un panel de origen de datos.
     dataSourcePane = new AS400JDBCDataSourcePane(myDataSource);

     // Cree un botón Aceptar.
     JButton okButton = new JButton("Aceptar");

     // Añada un objeto ActionListener al botón Aceptar. Al pulsar Aceptar,
     // se hará una llamada a applyChanges() para comprometer los
     // cambios efectuados en el origen de datos.
     okButton.addActionListener(new ActionListener()
            {
           public void actionPerformed(ActionEvent ev)
                {
                // Aplique todos los cambios realizados en la sección de
                // origen de datos al origen de datos. Si todos los cambios
                // se aplican bien, obtenga el origen de datos de la sección.
                   if (dataSourcePane.applyChanges())
                   {
                      System.out.println("Se ha pulsado Aceptar.");
                      myDataSource = dataSourcePane.getDataSource();
                      System.out.println(myDataSource.getServerName());
                   }
                }
             }
         );

     // Configure el marco para mostrar la sección y el botón Aceptar.
    frame.getContentPane().setLayout(new BorderLayout() );
frame.getContentPane ().add ("Center", dataSourcePane);
     frame.getContentPane ().add ("South", okButton);

     // Empaquete el marco.
     frame.pack ();

     //Visualice la sección y el botón Aceptar.
```

```
    frame.show ();
```

Ejemplo: cómo se utiliza VJobList para visualizar una lista de trabajos

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de lista de trabajos. Este programa presenta una lista de
trabajos
// una sección de panel explorador.
//
// Sintaxis de mandato:
//    VJobListExample sistema
//
// Este fuente es un ejemplo de "AS400ExplorerPane" de IBM Toolbox para
Java.
//
///////////////////////////////////////////////////////////////////////

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)
    {
        // De no haberse especificado un sistema, visualizar texto de ayuda
y
        // salir.
        if (args.length != 1)
        {
            System.out.println("Utilización:  VJobListExample sistema");
            return;
        }

                          try
        {
            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // Cree un objeto VJobList que representa la lista
            // de trabajos denominada QZDASOINIT.
          VJobList jobList = new VJobList (system);
jobList.setName ("QZDASOINIT");

            // Cree un marco.
            JFrame f = new JFrame ("Ejemplo de lista de trabajos");

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

    // Cree una sección de panel explorador para presentar la lista de
```

```
trabajos.            // Utilice load para cargar la información desde el
sistema.
    AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList);
explorerPane.addErrorListener (errorHandler);
                    explorerPane.load ();

            // Cuando el marco se cierre, salga.
            f.addWindowListener (new WindowAdapter () {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Diseñe el marco con la sección de panel explorador.
        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);
        }
    }


}
```

Ejemplo: cómo se utiliza un botón para llamar a un programa del servidor

Nota: lea la declaración de limitación de responsabilidad de ejemplos de código para obtener información legal importante.

```
///////////////////////////////////////////////////////////////////////
//
// Ejemplo de botón de llamada a programa. Este programa muestra cómo
// se utiliza un botón que llama a un programa del servidor. Intercambiará
datos
// con el programa del servidor mediante un parámetro de entrada y salida.
//
// Sintaxis de mandato:
//    ProgramCallButtonExample sistema
//
// Este fuente es un ejemplo de "ProgramCallButton" de IBM Toolbox para
Java.
//
///////////////////////////////////////////////////////////////////////

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;


    // Cree un objeto ProgramCallButtonExample y llame a la versión
    // no estática de main(). De no seguir este procedimiento,
    // las variables de clase (parm1, parm2, ...) deben declararse
    // estáticas (static). Si son estáticas, el escuchador de acciones
    // completadas no puede utilizarlas en Java 1.1.7 o 1.1.8.
    public static void main (String[] args)
    {
       ProgramCallButtonExample me = new ProgramCallButtonExample();
       me.Main(args);
    }

    public void Main (String[] args)
    {
        // De no haberse especificado un sistema, visualizar texto de ayuda
y
        // salir.
        if (args.length != 1)
        {
            System.out.println("Utilización:  ProgramCallButtonExample
sistema");
            return;
        }
```

```
                        try
        {
            // Cree un marco.
            JFrame f = new JFrame ("Ejemplo de botón de llamada a
programa");

            // Cree un adaptador de diálogo de errores. Así los errores
            // se mostrarán al usuario.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Cree un objeto AS400. El nombre del sistema se ha pasado
            // como primer argumento de línea de mandatos.
            AS400 system = new AS400 (args[0]);

            // Cree el nombre de vía de acceso del programa.
            QSYSObjectPathName programName = new QSYSObjectPathName ("QSYS",
                "QWCRSSTS", "PGM");

            // Cree un objeto ProgramCallButton. El botón tendrá
            // el texto "Renovar" y no tendrá ningún icono.
            ProgramCallButton button = new ProgramCallButton ("Renovar",
null);
            button.setSystem (system);
            button.setProgram (programName.getPath ());
            button.addErrorListener (errorHandler);

            // El primer parámetro es un parámetro de salida de 64 bytes.
            parm1 = new ProgramParameter (64);
            button.addParameter (parm1);

            // Usamos el segundo parámetro para establecer el tamaño de
            // almacenamiento intermedio del primero. Siempre lo
estableceremos
            // en 64. Recuerde que debemos convertir el valor int de Java
            // 64 al formato utilizado en el servidor.
     AS400Bin4 parm2Converter = new AS400Bin4 ();
            byte[] parm2Bytes = parm2Converter.toBytes (64);
            parm2 = new ProgramParameter (parm2Bytes);
            button.addParameter (parm2);

            // El tercer parámetro es el formato de estado. Siempre
            // utilizaremos "SSTS0200". Es un valor de tipo serie y
            // de nuevo debemos convertirlo al formato utilizado en el
servidor.
            AS400Text parm3Converter = new AS400Text (8, system);
            byte[] parm3Bytes = parm3Converter.toBytes ("SSTS0200");
            parm3 = new ProgramParameter (parm3Bytes);
            button.addParameter (parm3);

            // El cuarto parámetro es el parámetro de estadísticas de
reinicio.
            // Siempre pasaremos "*NO" como serie de 10 caracteres.
            AS400Text parm4Converter = new AS400Text (10, system);
            byte[] parm4Bytes = parm4Converter.toBytes ("*NO       ");
            parm4 = new ProgramParameter (parm4Bytes);
            button.addParameter (parm4);

            // El quinto parámetro es para información de error. Es un
            // parámetro de entrada/salida. No lo utilizaremos para
```

```
            // este ejemplo, pero debemos establecerlo en algún valor,
            // ya que si no el número de parámetros no coincidirá con
            // lo que espera el servidor.
            byte[] parm5Bytes = new byte[32];
            parm5 = new ProgramParameter (parm5Bytes, 0);
            button.addParameter (parm5);

            // Cuando se ejecute el programa, obtendremos un conjunto de
datos.
            // Necesitaremos una forma de mostrar esos datos al usuario.
            // En este caso, simplemente usaremos etiquetas sencillas y
            // campos de texto.
            JLabel cpuLabel = new JLabel ("Utilización de CPU: ");
            cpuField = new JTextField (10);
            cpuField.setEditable (false);

            JLabel dasdLabel = new JLabel ("Utilización de DASD: ");
            dasdField = new JTextField (10);
            dasdField.setEditable (false);

            JLabel jobsLabel = new JLabel ("Número de trabajos activos: ");
            jobsField = new JTextField (10);
            jobsField.setEditable (false);

            // Cuando el marco se cierre, salga.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Cuando se llame al programa, deberemos procesar la
            // información devuelta en el primer parámetro.
            // El formato de los datos del primer parámetro viene
            // documentado por el programa al que llamamos.
        button.addActionCompletedListener (new ActionCompletedListener ()
            {
             public void actionCompleted (ActionCompletedEvent event)
                {
                            try
                    {
                       // Obtenga los datos del primer parámetro.
                       // Tienen el formato del servidor.
                       byte[] parm1Bytes = parm1.getOutputData ();

                       // Cada fragmento de los datos que necesitamos
                       // es un valor int. Podemos crear un conversor
                       // para realizar todas las conversiones.
                       AS400Bin4 parm1Converter = new AS400Bin4 ();

                       // Obtenga la utilización de CPU empezando por el
byte 32.
                       // Establezca este valor en el campo de texto
correspondiente.
                       int cpu = parm1Converter.toInt (parm1Bytes, 32);
                       cpuField.setText (Integer.toString (cpu / 10) + "%");

                       // Obtenga la utilización de DASD empezando por el
```

```
byte 52.
                        // Establezca este valor en el campo de texto
correspondiente.
                        int dasd = parm1Converter.toInt (parm1Bytes, 52);
                        dasdField.setText (Integer.toString (dasd / 10000) +
"%");

                        // Obtenga el número de trabajos activos empezando
por el byte 36.
                        // Establezca este valor en el campo de texto
correspondiente.
                        int jobs = parm1Converter.toInt (parm1Bytes, 36);
                        jobsField.setText (Integer.toString (jobs));
                     }
                     catch (Exception e) { e.printStackTrace(); }
                  }
               });

               // Diseñe el marco.
               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);
            }
         }


}
```

Consejos para la programación

En este apartado se ofrecen diversos consejos que pueden ayudarle a utilizar IBM Toolbox para Java:

Cómo concluir el programa Java
Averigüe cómo concluir correctamente el programa Java.

Utilización de nombres de vía de acceso del sistema de archivos integrado
Lea información acerca de cómo utilizar los nombres de vía de acceso del sistema de archivos integrado en los programas. Esta sección trata sobre los nombres de vía de acceso del sistema de archivos integrado, los parámetros y los valores especiales.

Gestión de conexiones
Consulte cómo se debe utilizar la clase AS400 para iniciar y finalizar conexiones por socket, así como información sobre cómo cumplir la especificación Enterprise JavaBean.

Utilización de la máquina virtual Java (JVM) de OS/400
Aprenda a ejecutar las clases de IBM Toolbox para Java en la máquina virtual Java de OS/400. Esta sección explica cuál es la mejor forma de acceder a los recursos del servidor, cómo se ejecutan las clases y qué factores de inicio de sesión es preciso tener en cuenta.

»Conexión a una agrupación de almacenamiento auxiliar independiente (IASP)
Descubra cómo conectarse a una IASP. Una IASP es un conjunto de unidades de discos que puede activar o desactivar con independencia del resto del almacenamiento de un sistema.«

Manejo de errores al utilizar las clases de acceso de Toolbox para Java
Aprenda a utilizar las clases de excepción de Toolbox para Java para manejar los errores cuando utilice las clases de acceso de Toolbox para Java en un programa.

Manejo de errores al utilizar las clases de vaccess de Toolbox para Java
Consulte cómo utilizar las clases de eventos de error de Toolbox para Java para manejar los errores cuando utilice las clases de vaccess de Toolbox para Java en un programa.

Utilización de la clase Trace
Aprenda a utilizar la clase Trace para anotar los puntos de rastreo y mensajes de diagnóstico a fin de reproducir y diagnosticar los problemas en los programas.

Optimización de los programas
Aprenda a optimizar los programas para mejorar el rendimiento.

Obtención de un mejor rendimiento utilizando la máquina virtual Java de OS/400
Consulte cómo obtener un mejor rendimiento utilizando la máquina virtual Java de OS/400.

Gestión de las clases de Toolbox para Java en el cliente
Aprenda a utilizar la clase AS400ToolboxInstaller para gestionar las clases de IBM Toolbox para Java en el cliente.

Mejora del rendimiento de los archivos JAR
Consulte cómo utilizar las clases JarMaker de Toolbox para Java para crear archivos JAR de IBM Toolbox para Java más pequeños y rápidos de cargar.

Utilización del soporte de idioma nacional para Java
Consulte el tema relacionado con la utilización de IBM Toolbox para Java y el soporte de idioma nacional para Java.

Obtención de servicio y soporte
Utilice estos recursos para encontrar servicios de soporte para Toolbox para Java.

Cómo concluir el programa Java

Para asegurarse de que el programa concluye de manera adecuada, emita System.exit(0) como última instrucción antes de que finalice el programa Java.

Nota: evite utilizar System.exit(0) con servlets ya que ello concluye toda la máquina virtual Java.

IBM Toolbox para Java se conecta al servidor con hebras de usuario. Por ello, de no emitirse System.exit(0) puede que el programa Java no concluya debidamente.

Utilizar System.exit(0) no es un requisito imprescindible, sino una precaución. En ocasiones deberá utilizar este mandato para salir de un programa Java y, en cambio, no es problemático utilizar System.exit(0) cuando no es necesario.

Nombres de vía de acceso del sistema de archivos integrado para objetos de servidor

El programa Java debe utilizar nombres del sistema de archivos integrado para hacer referencia a objetos del servidor, como pueden ser programas, bibliotecas, mandatos o archivos en spool. El nombre de sistema de archivos integrado es el nombre de un objeto de servidor tal como se accedería a él en el sistema de archivos de biblioteca del sistema de archivos integrado en el servidor iSeries o AS/400e.

El nombre de vía de acceso puede constar de los siguientes componentes:

Componente del nombre de vía de acceso	Descripción
biblioteca	La biblioteca en la que reside el objeto. La biblioteca es una parte **obligatoria** de un nombre de vía de acceso del sistema de archivos integrado.El nombre de la biblioteca debe tener 10 caracteres como máximo, seguidos de **.lib**.
objeto	El nombre del objeto representado por el nombre de la vía de acceso del sistema de archivos integrado. El objeto es una parte **obligatoria** de un nombre de vía de acceso del sistema de archivos integrado.El nombre del objeto debe tener 10 caracteres como máximo, seguidos de **.tipo**, siendo **tipo** el tipo del objeto. Para encontrar los valores posibles de tipo, pulse la tecla Solicitud para el parámetro OBJTYPE en los mandatos CL (Control Language), como por ejemplo en el mandato Trabajar con objetos (WRKOBJ).
tipo	El tipo del objeto. El tipo se ha de especificar a la vez que el **objeto**.(Véase **objeto**, más arriba.)El nombre del tipo debe tener 6 caracteres como máximo.
miembro	El nombre del miembro representado por este nombre de vía de acceso del sistema de archivos integrado. El miembro es una parte **opcional** de un nombre de vía de acceso del sistema de archivos integrado.Únicamente puede especificarse cuando el **tipo de objeto** es **FILE**.El nombre del miembro debe tener 10 caracteres como máximo, seguidos de **.mbr**.

Al determinar y especificar el nombre del sistema de archivos integrado, tenga en cuenta estas condiciones:

- El carácter separador de la vía de acceso es la barra inclinada hacia delante (/).
- El directorio de nivel raíz, llamado QSYS.LIB, contiene la estructura de bibliotecas del servidor.
- Los objetos que residen en la biblioteca QSYS del servidor tienen este formato:

```
/QSYS.LIB/objeto.tipo
```

- Los objetos que residen en otras bibliotecas tienen este formato:

```
/QSYS.LIB/biblioteca.LIB/objeto.tipo
```

- La extensión del tipo de objeto es la abreviatura del servidor que se utiliza para ese tipo de objeto.

Si desea ver una lista de estos tipos, entre un mandato CL que tenga el parámetro de tipo de objeto y pulse **F4** (Solicitud) para obtener los valores de tipo. Por ejemplo, el mandato Trabajar con objetos (WRKOBJ) tiene un parámetro de tipo de objeto.

La tabla siguiente muestra una lista de los tipos de objeto más utilizados y la abreviatura correspondiente a cada tipo:

Tipo de objeto	Abreviatura
mandato	.CMD
cola de datos	.DTAQ

archivo	.FILE
recurso de fonts	.FNTRSC
definición de formulario	.FORMDF
biblioteca	.LIB
miembro	.MBR
preformato	.OVL
definición de página	.PAGDFN
segmento de página	.PAGSET
programa	.PGM
cola de salida	.OUTQ
archivo en spool	.SPLF

Las descripciones siguientes pueden ayudarle a determinar cómo se especifican los nombres de vía de acceso del sistema de archivos integrado:

Nombre de sistema de archivos integrado	Descripción
/QSYS.LIB/MI_BIBL.LIB/MI_PROG.PGM	Programa MI_PROG de la biblioteca MI_BIBL en el servidor
/QSYS.LIB/MI_BIBL.LIB/MI_COLA.DTAQ	Cola de datos MI_COLA de la biblioteca MI_BIBL en el servidor
/QSYS.LIB/AÑO1998.LIB/MES.FILE/JULIO.MBR	Miembro JULIO del archivo MES de la biblioteca AÑO1998 en el servidor

Valores especiales del sistema de archivos integrado

Diversas clases de IBM Toolbox para Java reconocen valores especiales en los nombres de vía de acceso del sistema de archivos integrado. El formato tradicional de estos valores especiales (tal como se utilizan en una línea de mandatos de iSeries) empieza con un asterisco (***ALL**). Sin embargo, en un programa Java que utiliza las clases de Toolbox para Java, el formato de estos valores especiales empieza y termina con signos de porcentaje (**%ALL%**).

Nota: en el sistema de archivos integrado, un asterisco es un carácter comodín.

La tabla siguiente muestra cuáles de estos valores especiales reconocen las clases de IBM Toolbox para Java para componentes específicos del nombre de vía de acceso. Asimismo, la tabla indica cómo varía el formato tradicional de estos valores especiales del formato empleado en las clases de Toolbox para Java.

Componente del nombre de vía de acceso	Formato tradicional	Formato de Toolbox para Java
Nombre de biblioteca	*ALL	%ALL%
	*ALLUSR	%ALLUSR%
	*CURLIB	%CURLIB%
	*LIBL	%LIBL%
	*USRLIBL	%USRLIBL%
Nombre de objeto	*ALL	%ALL%
Nombre de miembro	*ALL	%ALL%
	*FILE	%FILE%
	*FIRST	%FIRST%
	*LAST	%LAST%

En la clase QSYSObjectPathName encontrará información sobre cómo se construyen y analizan los nombres del sistema de archivos integrado.

Para obtener más información sobre los conceptos del sistema de archivos integrado, consulte los conceptos del sistema de archivos integrado.

Gestión de conexiones

Es importante poder crear, iniciar y finalizar conexiones con el servidor. La información siguiente describe conceptos fundamentales para la gestión de las conexiones con el servidor y facilita también algunos ejemplos de código.

Para conectarse a un sistema iSeries, el programa Java debe crear un objeto AS400. El objeto AS400 contiene como máximo una conexión por socket para cada tipo de servidor iSeries. Un servicio corresponde a un trabajo del servidor y hace de interfaz con los datos del servidor.

Nota:cuando cree Enterprise JavaBeans (EJB), debe cumplir la especificación EJB que no permite hebras durante la conexión. Aunque al desactivar el soporte para hebras de IBM Toolbox para Java la aplicación puede ir más lenta, es necesario cumplir la especificación EJB.

Toda conexión con cada uno de los servidores tiene su propio trabajo en el iSeries. Hay un servidor distinto para cada uno de estos elementos:

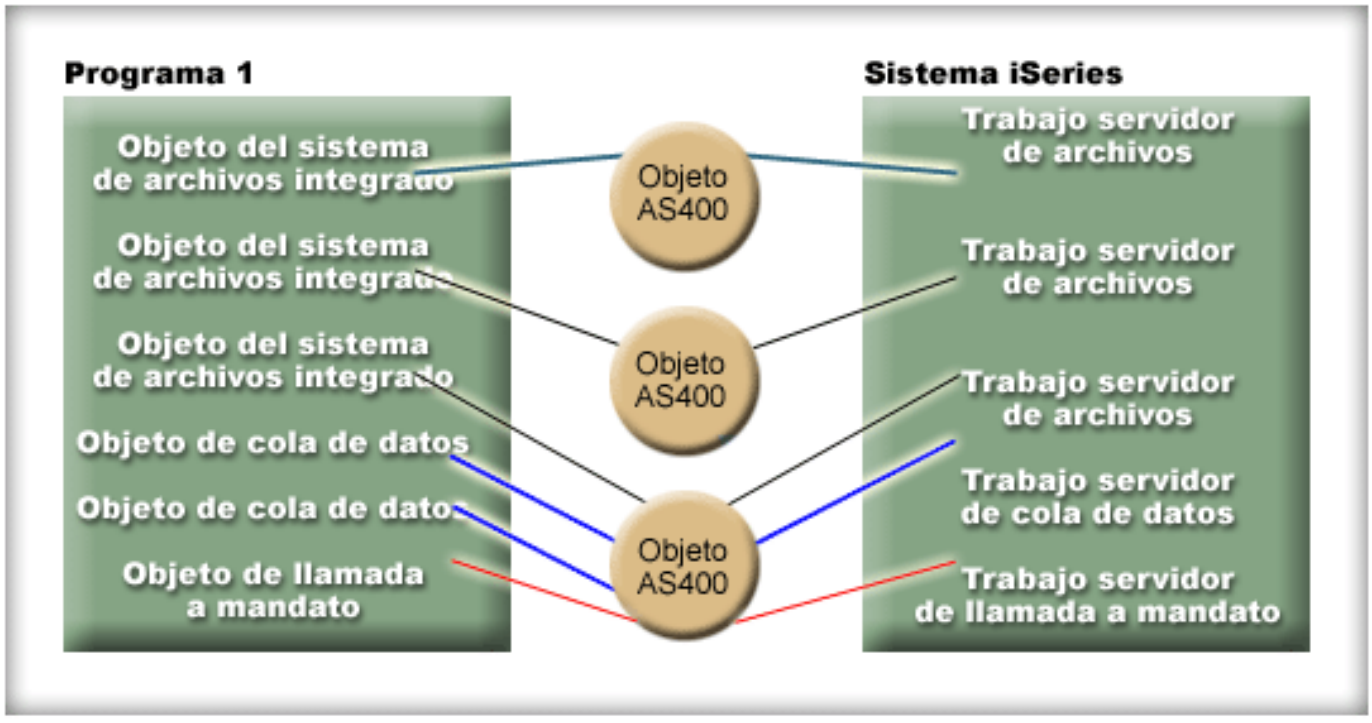
- JDBC
- Llamada a programa y llamada a mandato
- Sistema de archivos integrado
- Imprimir
- Cola de datos
- Acceso a nivel de registro

Notas:

- Las clases de impresión utilizan una conexión por socket por objeto AS400 si la aplicación no intenta realizar dos tareas que requieran el servidor de impresión de red a la vez.
- Una clase de impresión crea, de ser necesario, conexiones por socket adicionales con el servidor de impresión de red. Las conversaciones adicionales se desconectan si no se utilizan durante 5 minutos.

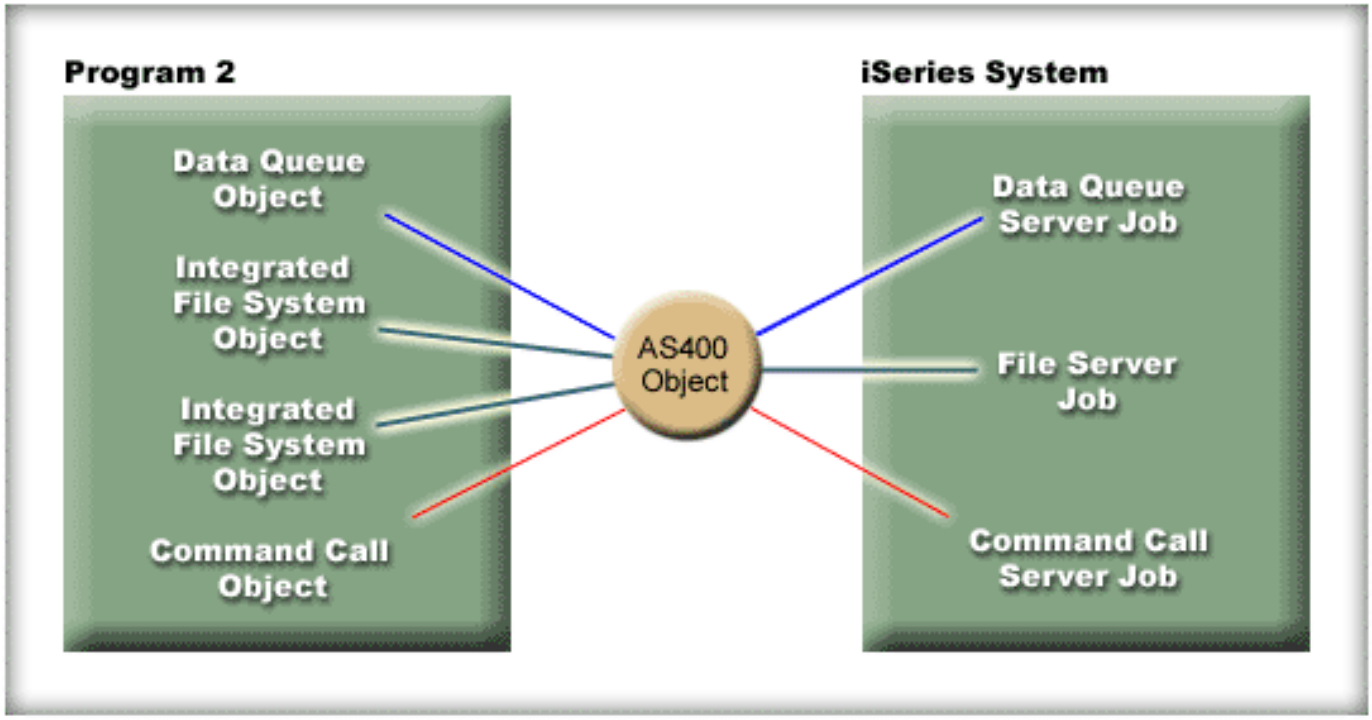
El programa Java puede controlar el número de conexiones con iSeries. Para optimizar el rendimiento de las comunicaciones, un programa Java puede crear varios objetos AS400 para el mismo sistema, tal como se muestra en la figura 1. De este modo se crean varias conexiones por socket con el iSeries.

Figura 1: programa Java que crea múltiples objetos AS400 y conexiones por socket para el mismo sistema iSeries



Para conservar los recursos del sistema iSeries, cree únicamente un objeto AS400 tal como se muestra en la figura 2. Este enfoque reduce el número de conexiones, lo que a su vez reduce la cantidad de recursos utilizados en el sistema.

Figura 2: programa Java que crea un solo objeto AS400 y una única conexión por socket para el mismo sistema iSeries



Nota: aunque al crear más conexiones aumenta la cantidad de recursos utilizados en el sistema, el hecho de crear más conexiones puede aportar una ventaja. Al tener más conexiones, el programa Java puede efectuar varios procesos en paralelo, lo que permite conseguir una mejor productividad (número de transacciones por segundo) y una mayor velocidad de la aplicación.

También puede elegir utilizar una agrupación de conexiones para gestionar las conexiones, como se muestra en la figura 3. Este enfoque reduce la cantidad de tiempo que se tarda en conectarse a iSeries al reutilizar una conexión establecida anteriormente para el usuario.

Figura 3: programa Java que obtiene una conexión de un objeto AS400ConnectionPool a un servidor iSeries

Los ejemplos que hay a continuación muestran cómo se crean y utilizan los objetos AS400:

Ejemplo 1: en este ejemplo se crean dos objetos CommandCall que envían mandatos al mismo sistema iSeries. Como los objetos CommandCall utilizan el mismo objeto AS400, solamente se crea una conexión con el sistema.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree dos objetos llamada a mandato que utilicen
    // el mismo objeto AS400.
CommandCall cmd1 = new CommandCall(sys,"myCommand1");
CommandCall cmd2 = new CommandCall(sys,"myCommand2");

    // Ejecute los mandatos. Se realiza una conexión al
    // ejecutarse el primer mandato. Como los dos mandatos
    // emplean el mismo objeto AS400, el segundo objeto mandato
    // empleará la conexión establecida por el primero.
cmd1.run();
cmd2.run();
```

Ejemplo 2: en este ejemplo se crean dos objetos CommandCall que envían mandatos al mismo sistema iSeries. Como los objetos CommandCall utilizan distintos objetos AS400, se crean dos conexiones con el sistema.

```
                // Cree dos objetos AS400 para el mismo servidor.
AS400 sys1 = new AS400("mySystem.myCompany.com");
AS400 sys2 = new AS400("mySystem.myCompany.com");

    // Cree dos objetos llamada a mandato. Utilizan
    // distintos objetos AS400.
CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

    // Ejecute los mandatos. Se realiza una conexión al
    // ejecutarse el primer mandato. Como el segundo objeto
    // mandato usa un objeto AS400 distinto, se realiza una
    // segunda conexión al ejecutarse el segundo mandato.
cmd1.run();
cmd2.run();
```

Ejemplo 3: en este ejemplo se crean un objeto CommandCall y un objeto IFSFileInputStream que utilizan un mismo objeto AS400. Como el objeto CommandCall y el objeto IFSFileInputStream emplean distintos servicios del sistema iSeries, se

crean dos conexiones.

```
        // Cree un objeto AS400.
     AS400 newConn1 = new AS400("mySystem.myCompany.com");

        // Cree un objeto llamada a mandato.
     CommandCall cmd = new CommandCall(newConn1,"myCommand1");

        // Cree el objeto archivo. Crearlo hace que el
        // objeto AS400 se conecte al servicio de archivos.
     IFSFileInputStream file = new IFSFileInputStream(newConn1,"/myfile");

        // Ejecute el mandato. Se realiza una conexión con
        // el servicio de mandatos al ejecutarse el mandato.
     cmd.run();
```

Ejemplo 4: en este ejemplo se utiliza un objeto AS400ConnectionPool para obtener una conexión de iSeries. Este ejemplo (como el ejemplo 3 anterior) no especifica ningún servicio, por lo que la conexión con el servicio de mandatos se efectúa cuando se ejecuta el mandato.

```
        // Cree una AS400ConnectionPool.
     AS400ConnectionPool testPool1 = new AS400ConnectionPool();
        // Cree una conexión.
     AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID",
"myPassword");
                       // Cree un objeto de llamada a mandato que utilice el
objeto AS400.
     CommandCall cmd = new CommandCall(newConn1,"myCommand1");
        // Ejecute el mandato. Se realiza una conexión con
        // el servicio de mandatos al ejecutarse el mandato.
     cmd.run();
                       // Devuelva la conexión a la agrupación.
     testPool1.returnConnectionToPool(newConn1);
```

Ejemplo 5: en este ejemplo se utiliza AS400ConnectionPool para conectarse a un servicio concreto al solicitar la conexión de la agrupación. Esto elimina el tiempo necesario para conectarse al servicio cuando se ejecuta el mandato (véase el ejemplo 4 anterior). Si la conexión se devuelve a la agrupación, la siguiente llamada para obtener una conexión puede devolver el mismo objeto de conexión. Esto significa que no es necesario ningún tiempo de conexión adicional, ni de creación ni de uso.

```
        // Cree una AS400ConnectionPool.
     AS400ConnectionPool testPool1 = new AS400ConnectionPool();
                       // Cree una conexión con el servicio AS400.COMMAND.
(Emplee las constantes de número de servicio
                       // definidas en la clase AS400 (FILE, PRINT, COMMAND,
DATAQUEUE, etc.))
     AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID",
"myPassword", AS400.COMMAND);
                       // Cree un objeto de llamada a mandato que utilice el
objeto AS400.
     CommandCall cmd = new CommandCall(newConn1,"myCommand1");
                       // Ejecute el mandato. Ya se ha realizado una
conexión
                       // con el servicio de mandatos.
     cmd.run();
                       // Devuelva la conexión a la agrupación.
     testPool1.returnConnectionToPool(newConn1);
                       // Obtenga otra conexión con el servicio de mandatos.
En este caso, devolverá la misma
                       // conexión que anteriormente, con lo que se evita el
tiempo de conexión adicional tanto
                       // ahora como al utilizarse el servicio de mandatos.
     AS400 newConn2 = testPool1.getConnection("myAS400", "myUserID",
```

```
"myPassword", AS400.COMMAND);
```

Inicio y finalización de las conexiones

El programa Java puede controlar cuándo se inicia una conexión y cuándo se finaliza. Por omisión, el inicio de una conexión se lleva a cabo en el momento en que se necesita información del servidor. Para controlar exactamente cuándo se establece la conexión, puede efectuar una llamada al método connectService() en el objeto AS400 para preconectarse al servidor.

Con un objeto AS400ConnectionPool, puede crear una conexión preconectada a un servicio sin efectuar ninguna llamada al método connectService(), como se muestra en el ejemplo 5 anterior.

Los ejemplos que figuran a continuación muestran programas Java que se conectan al iSeries y se desconectan de él.

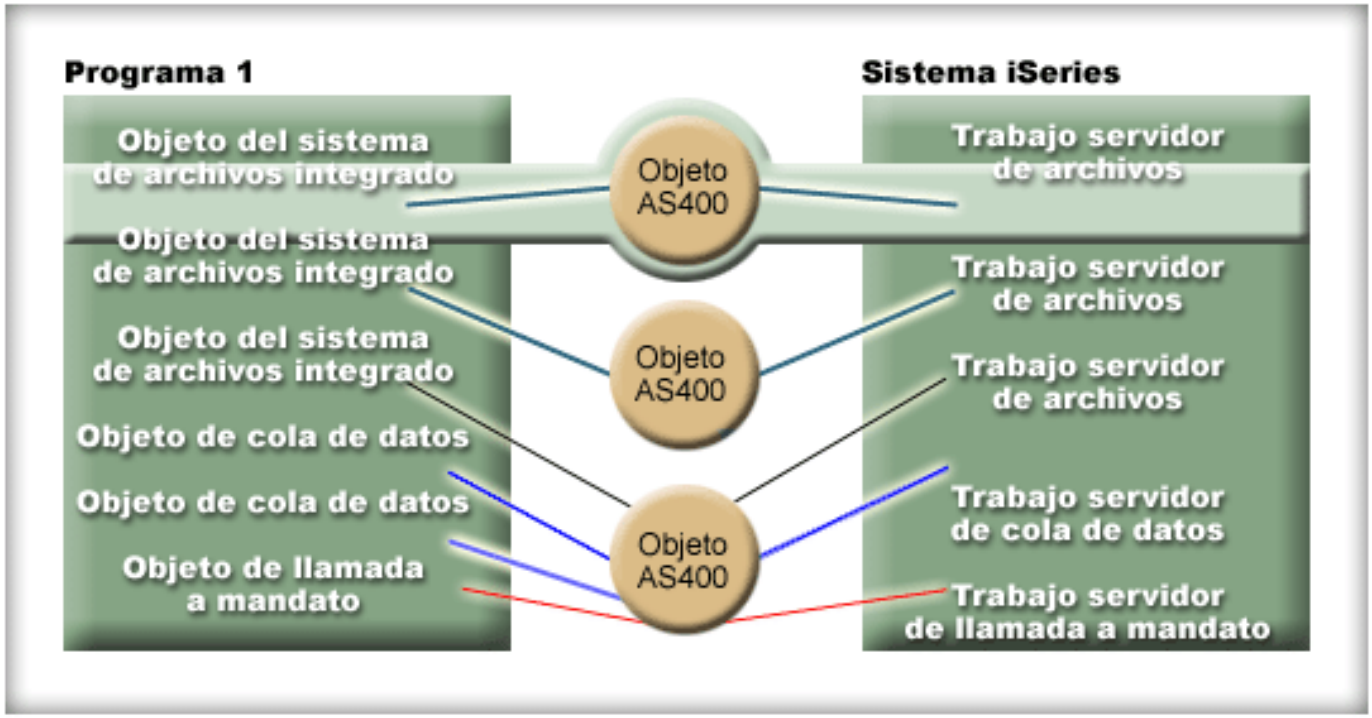
Ejemplo 1: este ejemplo muestra cómo se hace la preconexión con el iSeries:

```
    // Cree un objeto AS400.
AS400 system1 = new AS400("mySystem.myCompany.com");

    // Conéctese al servicio de mandatos. Hágalo ahora, en
    // lugar de en el momento de enviar los datos al servicio
    // de mandatos por primera vez. Es una acción opcional,
    // pues el objeto AS400 se conectará cuando sea necesario.
system1.connectService(AS400.COMMAND);
```

Ejemplo 2: una vez iniciada una conexión, el programa Java es responsable de la desconexión, efectuándola implícitamente el objeto AS400, o explícitamente el propio programa Java. Un programa Java efectúa la desconexión realizando una llamada al método disconnectService() en el objeto AS400. Para aumentar el rendimiento, el programa Java solo debería desconectar cuando haya terminado de utilizar un servicio. Si el programa Java se desconecta antes de haber terminado de utilizar un servicio, el objeto AS400 se reconecta (si es que es posible la reconexión) cuando se necesitan datos del servicio.

La figura 4 muestra cómo la acción de desconectar la conexión correspondiente a la primera conexión del objeto de sistema de archivos integrado sólo finaliza esa instancia individual de la conexión del objeto AS400, no todas las conexiones de objeto de sistema de archivos integrado.

Figura 4: se desconecta un objeto individual que utiliza su propio servicio de una instancia de un objeto AS400

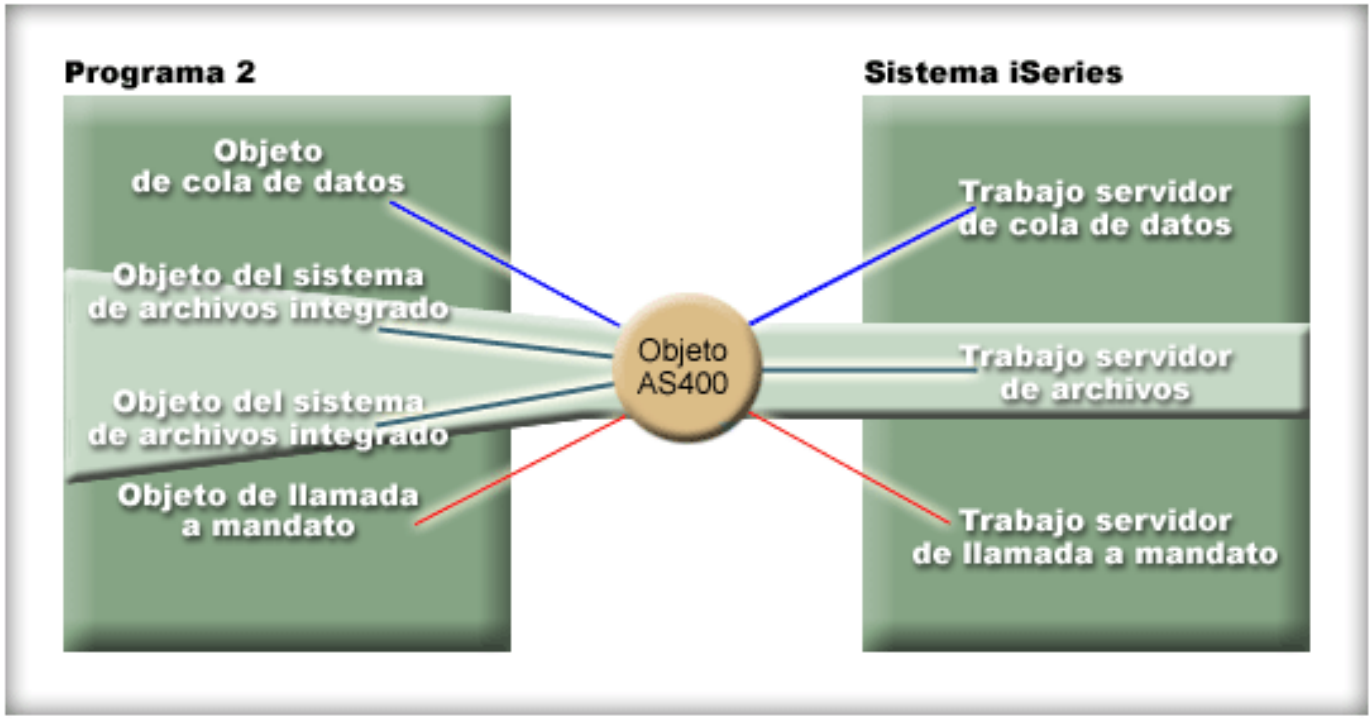


Este ejemplo muestra cómo el programa Java desconecta una conexión:

```
    // Cree un objeto AS400.
AS400 system1 = new AS400("mySystem.myCompany.com");

    // ...Utilice la llamada a mandato para enviar varios mandatos
    // al servidor. Debido a que no se ha llamado a connectService(),
    // el objeto AS400 se conecta automáticamente al ejecutarse el
    // el primer mandato.

    // El hecho de terminar de enviar todos los mandatos hace que
    // se pueda desconectar la conexión.
system1.disconnectService(AS400.COMMAND);
```

Ejemplo 3: varios objetos que utilizan el mismo servicio y comparten el mismo objeto AS400 comparten una conexión. La desconexión finaliza la conexión para todos los objetos que utilizan el mismo servicio de cada instancia de un objeto AS400, como se muestra en la figura 5.

Figura 5: se desconectan todos los objetos que utilizan el mismo servicio de una instancia de un objeto AS400



Por ejemplo, supongamos que dos objetos CommandCall utilizan el mismo objeto AS400. Cuando se llama a disconnectService(), la conexión finaliza para los dos objetos CommandCall. Cuando se llama al método run() para el segundo objeto CommandCall, el objeto AS400 debe reconectarse al servicio:

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree dos objetos llamada a mandato.
CommandCall cmd1 = new CommandCall(sys,"myCommand1");
CommandCall cmd2 = new CommandCall(sys,"myCommand2");

    // Ejecute el primer mandato.
cmd1.run();

    // Desconéctese del servicio de mandatos.
sys.disconnectService(AS400.COMMAND);
```

```
    // Ejecute el segundo mandato. El objeto AS400
    // debe volver a conectarse al servidor.
cmd2.run();

    // Desconéctese del servicio de mandatos. Éste
    // es el lugar correcto para desconectar.
sys.disconnectService(AS400.COMMAND);
```

Ejemplo 4: no todas las clases de IBM Toolbox para Java se reconectan automáticamente. Algunas llamadas a método en las clases del sistema de archivos integrado no se reconectan porque el archivo puede haber cambiado. Mientras el archivo estaba desconectado, algún otro proceso puede haber suprimido el archivo o cambiado su contenido. En este ejemplo hay dos objetos archivo que utilizan el mismo objeto AS400. Cuando se llama a disconnectService(), la conexión finaliza para los dos objetos archivo. El método read() para el segundo objeto IFSFileInputStream falla porque ha dejado de tener una conexión con el servidor.

```
    // Cree un objeto AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

    // Cree dos objetos archivo. Se crea una conexión
    // con el servidor cuando se crea el primer objeto.
    // El segundo objeto utiliza la conexión creada
    // por el primer objeto.
IFSFileInputStream file1 = new IFSFileInputStream(sys,"/file1");
IFSFileInputStream file2 = new IFSFileInputStream(sys,"/file2");

    // Lea en el primer archivo y, después, ciérrelo.
int i1 = file1.read();
file1.close();

    // Desconéctese del servicio de archivos.
sys.disconnectService(AS400.FILE);

    // Intente leer en el segundo archivo. Este intento
    // falla debido a que ha dejado de existir la conexión
    // con el servicio de archivos. El programa debería
    // haber desconectado más tarde o hacer que el segundo
    // archivo utilizase otro objeto AS400 (y así el
    // archivo tendría su propia conexión).
int i2 = file2.read();

    // Cierre el segundo archivo.
file2.close();

    // Desconéctese del servicio de archivos. Éste
    // es el lugar correcto para desconectar.
sys.disconnectService(AS400.FILE);
```

Máquina virtual Java de OS/400

Las clases de IBM Toolbox para Java se ejecutan en la máquina virtual Java (JVM) de IBM Developer Kit para Java (OS/400). En realidad, las clases se ejecutan en cualquier plataforma que dé soporte a las especificaciones JDK (Java Development Kit) 1.1.x y J2SDK (Java 2 Software Development Kit).

Cuando ejecute las clases de IBM Toolbox para Java en la máquina virtual Java de OS/400, realice estas tareas:

- Elija si va a utilizar la máquina virtual Java de OS/400 o las clases de IBM Toolbox para Java para acceder a los recursos del servidor iSeries al llevar a cabo la ejecución en la máquina virtual Java de OS/400.
- Consulte la ejecución de clases de IBM Toolbox para Java en la máquina virtual Java de OS/400.
- Consulte cómo establecer el nombre del sistema, el ID de usuario y la contraseña en la máquina virtual Java de OS/400.

Para obtener más información acerca del soporte del servidor iSeries para las distintas plataformas Java, consulte Soporte para varios JDK en IBM Developer Kit para Java.

Comparación de la máquina virtual Java de OS/400 y las clases de IBM Toolbox para Java

Siempre dispone de al menos dos maneras de acceder a un recurso del servidor iSeries cuando el programa Java se ejecuta en la máquina virtual Java (JVM) de IBM Developer Kit para Java (OS/400). Puede utilizar una de estas dos interfaces:

- Servicios integrados en Java
- Una clase de IBM Toolbox para Java

A la hora de decidir qué interfaz debe utilizar, tenga en cuenta estos factores:

- **Ubicación** - El lugar de la ejecución del programa es el factor más importante a la hora de decidir qué conjunto de interfaces se ha de utilizar.El programa:
  - ¿Se ejecuta sólo en el cliente?
  - ¿Se ejecuta sólo en el servidor?
  - ¿Se ejecuta en el cliente y en el servidor, pero el recurso es en los dos casos un recurso del servidor iSeries?
  - ¿Se ejecuta en una máquina virtual Java de OS/400 y accede a los recursos de otro servidor iSeries?
  - ¿Se ejecuta en servidores de distinto tipo?

  Si el programa se ejecuta en el cliente y en el servidor (incluido el caso en que un servidor iSeries sea cliente de un segundo servidor iSeries) y sólo accede a recursos del servidor iSeries, puede ser mejor utilizar las interfaces de IBM Toolbox para Java.

  Si el programa debe acceder a datos existentes en servidores de muchos tipos, puede ser mejor utilizar las interfaces Java nativas.
- **Coherencia / Portabilidad** - La capacidad de ejecutar clases de IBM Toolbox para Java en servidores iSeries significa que pueden utilizarse las mismas interfaces para los programas clientes y los programas servidores.Cuando únicamente se tiene que aprender cómo funciona una sola interfaz para los programas clientes y los programas servidores, se puede ser más productivo.

  Sin embargo, escribir en interfaces de IBM Toolbox para Java hace que el programa sea menos portable entre **servidores**.

  Si el programa debe ejecutarse en un servidor iSeries además de en otros servidores, tal vez sea mejor utilizar los servicios integrados en Java.
- **Complejidad** - La interfaz de IBM Toolbox para Java se ha construido especialmente para proporcionar un acceso fácil a un recurso del servidor iSeries. Con frecuencia, el uso de la interfaz de IBM Toolbox para Java tiene como única alternativa la escritura de un programa que acceda al recurso y se comunique con ese programa mediante JNI (interfaz Java nativa).

  Debe decidir qué es más importante: gozar de una mayor neutralidad Java y escribir un programa para acceder al recurso, o utilizar la interfaz de IBM Toolbox para Java, que es menos portable.
- **Funciones** - La interfaz de IBM Toolbox para Java suele proporcionar más funciones que la interfaz de Java.Por ejemplo, la clase IFSFileOutputStream del programa bajo licencia IBM Toolbox para Java tiene más funciones que la clase FileOutputStream de java.io. No obstante, el uso de IFSFileOutputStream hace que el programa sea específico de los servidores iSeries. El uso de la clase de IBM Toolbox para Java hace que sea menor la portabilidad entre **servidores**.

  Debe decidir si es más importante la portabilidad o si prefiere beneficiarse de las funciones adicionales.
- **Recursos** - Al ejecutarse en la máquina virtual Java de OS/400, muchas de las clases de IBM Toolbox para Java siguen efectuando peticiones mediante los servidores de sistema principal. Por lo tanto, hay un segundo trabajo (el trabajo servidor) que lleva a cabo la petición de acceder a un recurso.

Esta petición puede emplear más recursos que una interfaz Java nativa que se ejecute bajo el trabajo del programa Java.

- **Servidor iSeries como cliente** - Si el programa se ejecuta en un servidor iSeries y accede a los datos existentes en un segundo servidor iSeries, la mejor solución sería utilizar las clases de IBM Toolbox para Java. Estas clases proporcionan un acceso fácil al recurso existente en el segundo servidor iSeries.

  Ejemplo de ello es el acceso a las colas de datos. Las interfaces de DataQueue del programa bajo licencia IBM Toolbox para Java proporcionan un acceso fácil al recurso de cola de datos.

  El uso de IBM Toolbox para Java también significa que el programa funciona tanto en un cliente como en un servidor para acceder a una cola de datos de un servidor iSeries. También funciona cuando se ejecuta en un servidor iSeries para acceder a una cola de datos de otro servidor iSeries.

  La alternativa es escribir un programa aparte (en C, por ejemplo) que acceda a la cola de datos. El programa Java efectúa una llamada a este programa cuando necesita acceder a la cola de datos.

  Este método supone una mayor portabilidad entre servidores; se puede tener un programa Java que maneje el acceso a cola de datos y distintas versiones del programa para cada uno de los servidores soportados.

Ejecución de clases de IBM Toolbox para Java en la máquina virtual Java de OS/400

A continuación figuran algunas consideraciones especiales que deben tenerse en cuenta al ejecutar las clases de IBM Toolbox para Java en la máquina virtual Java (JVM) de IBM Developer Kit para Java (OS/400):

JDBC

Para los programas que se ejecutan en la máquina virtual Java de OS/400, están disponibles dos controladores JDBC suministrados por IBM:

- El controlador JDBC de IBM Toolbox para Java
- El controlador JDBC de IBM Developer Kit para Java

Es mejor utilizar el controlador JDBC de IBM Toolbox para Java cuando el programa se ejecuta en un entorno de cliente/servidor.

Es mejor utilizar el controlador JDBC de IBM Developer Kit para Java cuando el programa se ejecuta en un servidor iSeries.

Si un mismo programa se ejecuta en la estación de trabajo y en el servidor, debe cargar el controlador correcto mediante una propiedad del sistema en vez de codificar el nombre del controlador en el programa.

Llamada a programa

Una llamada a programa se puede realizar de estas dos maneras comunes:

- Con la clase ProgramCall de IBM Toolbox para Java
- Mediante una llamada JNI (interfaz Java nativa)

La clase ProgramCall del programa bajo licencia IBM Toolbox para Java posee la ventaja de que puede efectuar llamadas a cualquier programa del servidor iSeries.

Mediante JNI tal vez no pueda efectuar una llamada al programa del servidor iSeries. Una ventaja de JNI es que ofrece más portabilidad entre plataformas de servidor.

Llamada a mandato

Una llamada a mandato se puede realizar de estas dos maneras comunes:

- Con la clase CommandCall de IBM Toolbox para Java
- Mediante java.lang.runtime.exec()

La clase CommandCall genera una lista de los mensajes que están disponibles para el programa Java una vez completado el mandato. Mediante java.lang.runtime.exec(), la lista de mensajes no está disponible.

java.lang.runtime.exec() es portable entre muchas plataformas, de modo que si el programa debe acceder a archivos existentes en servidores de distintos tipos, la mejor solución es java.lang.runtime.exec().

Sistema de archivos integrado

Formas comunes de acceder a un archivo del sistema de archivos integrado del servidor iSeries:

- Con las clases IFSFile del programa bajo licencia IBM Toolbox para Java
- Con las clases archivo que forman parte de java.io

Las clases del sistema de archivos integrado de IBM Toolbox para Java poseen la ventaja de que proporcionan más funciones que las clases de java.io. Las clases de IBM Toolbox para Java también funcionan en los applets y no necesitan un método de redirección (como iSeries Access para Windows) para acceder al servidor desde una estación

de trabajo.

Las clases de java.io son portables entre muchas plataformas, lo que es una ventaja. Si el programa debe acceder a archivos existentes en servidores de distintos tipos, java.io es una solución mejor.

Si utiliza las clases de java.io en un cliente, necesitará un método de redirección (como iSeries Access para Windows) para acceder al sistema de archivos del servidor.

Establecer el nombre del sistema, el ID de usuario y la contraseña con un objeto AS400 en la máquina virtual Java de OS/400

El objeto AS400 admite valores especiales para el nombre del sistema, el ID de usuario y la contraseña cuando el programa Java se ejecuta en la máquina virtual Java (JVM) de IBM Developer Kit para Java (OS/400).

Cuando ejecute un programa en la máquina virtual Java de OS/400, tenga en cuenta algunos valores especiales y otras consideraciones:

- La solicitud de ID de usuario y contraseña queda inhabilitada cuando el programa se ejecuta en el servidor. Para obtener más información acerca de los valores de ID de usuario y contraseña en el entorno servidor, véase Resumen de los valores de ID de usuario y contraseña en un objeto AS400.
- Si no se ha establecido el nombre del sistema, el ID de usuario o la contraseña en el objeto AS400, éste se conecta al servidor actual mediante el ID de usuario y la contraseña del trabajo que inició el programa Java. **Es preciso suministrar una contraseña cuando se utiliza el acceso a nivel de registro mientras se esté conectando a máquinas cuyo release sea igual o anterior a v4r3.Al conectarse a una máquina cuyo release sea igual o posterior a v4r4, el objeto AS400 puede propagar la contraseña del usuario conectado al igual que el resto de los componentes de IBM Toolbox para Java.**
- Como nombre del sistema, se puede utilizar el valor especial **localhost**.En este caso, el objeto AS400 se conecta al servidor actual.
- El valor especial, ***current**, puede utilizarse como ID de usuario o contraseña en el objeto AS400. En este caso, se utiliza el ID de usuario o la contraseña (o ambos) del trabajo que inició el programa Java.Para obtener más información sobre *current, consulte la información que se facilita a continuación en el apartado Notas.
- El valor especial, ***current**, puede utilizarse como ID de usuario o contraseña en el objeto AS400 cuando el programa Java se ejecuta en la máquina virtual Java de OS/400 de un servidor iSeries, y el programa accede a los recursos existentes en otro servidor iSeries. En este caso, al conectarse al sistema destino se utiliza el ID de usuario y la contraseña del trabajo que inició el programa Java en el sistema origen. Para obtener más información sobre *current, consulte la información que se facilita a continuación en el apartado Notas.

  Notas:

  - El programa no puede establecer la contraseña en "*current" si se utiliza el acceso a nivel de registro y un release igual o anterior a V4R3. Cuando se utiliza el acceso a nivel de registro, "localhost" es válido para el nombre del sistema, y "*current" es válido para el ID de usuario; sin embargo, el programa Java debe suministrar la contraseña.
  - El valor *current sólo funciona en los sistemas que se ejecutan en la Versión 4 Release 3 (V4R3) y posteriores. En los sistemas que se ejecutan en la V4R2, es preciso especificar la contraseña y el ID de usuario.

Los ejemplos que hay a continuación muestran cómo se utiliza el objeto AS400 con la máquina virtual Java de OS/400.

Ejemplo 1: un programa Java, cuando se ejecuta en la máquina virtual Java de OS/400, no tiene que suministrar un nombre de sistema, un ID de usuario o una contraseña.

Cuando se utiliza el acceso a nivel de registro, es preciso suministrar una contraseña.

Si no se suministran estos valores, el objeto AS400 se conecta al sistema local utilizando el ID de usuario y la contraseña del trabajo que inició el programa Java.

Cuando el programa se ejecuta en la máquina virtual Java de OS/400, establecer el nombre del sistema en **localhost** equivale a no establecer el nombre del sistema. El siguiente ejemplo muestra cómo conectarse al servidor actual:

```
// Cree dos objetos AS400. Si el programa Java se ejecuta en la
```

```
// JVM de OS/400, el comportamiento de ambos objetos es el mismo.
// Se conectarán al servidor actual con el ID de usuario y la
// contraseña del trabajo que inició el programa Java.
AS400 sys  = new AS400()
AS400 sys2 = new AS400("localhost")
```

Ejemplo 2: el programa Java puede establecer un ID de usuario y una contraseña incluso cuando se ejecuta en la máquina virtual Java de OS/400. Estos valores alteran temporalmente el ID de usuario y la contraseña del trabajo que inició el programa Java.

En el siguiente ejemplo, el programa Java se conecta al servidor actual, pero utiliza un ID de usuario y una contraseña distintos de los del trabajo que inició el programa Java.

```
// Cree un objeto AS400. Conéctese al servidor actual pero no
// utilice el ID de usuario y la contraseña del trabajo que inició
// el programa. Se emplean los valores suministrados.
AS400 sys = new AS400("localhost", "USR2", "PSWRD2")
```

Ejemplo 3: un programa Java que se ejecuta en un servidor puede conectarse a otros sistemas iSeries y utilizar sus recursos.

Si se indica ***current** para el ID de usuario y la contraseña, cuando el programa Java se conecte al servidor destino se utilizará el ID de usuario y la contraseña del trabajo que inició el programa Java.

En el ejemplo que figura a continuación, el programa Java se ejecuta en un servidor, pero utiliza los recursos de otro servidor. Cuando el programa se conecta al segundo servidor, se utiliza el ID de usuario y la contraseña del trabajo que inició el programa Java.

```
// Cree un objeto AS400. Este programa se ejecutará en un servidor
// pero se conectará a un segundo servidor (denominado "destino").
// Como se utiliza *current para el ID de usuario y la
// contraseña, se emplearán el ID de usuario y la contraseña del
// trabajo que inició el programa al conectarse al segundo servidor.
AS400 target = new AS400("target", "*current", "*current")
```

Agrupación de almacenamiento auxiliar independiente (IASP)

Una agrupación de almacenamiento auxiliar independiente (IASP) es un conjunto de unidades de discos que puede activar o desactivar con independencia del resto del almacenamiento de un sistema. Las IASP pueden contener cualquiera de los elementos siguientes:

- Uno o varios sistemas de archivos definidos por el usuario
- Una o varias bibliotecas externas

Cada IASP contiene toda la información del sistema necesaria asociada a los datos que posee. De este modo, mientras el sistema está activo, puede desactivar la IASP, activarla o conmutar entre sistemas.

Para obtener más información, consulte la información acerca de las ASP independientes y las ASP de usuario.

Puede emplear la propiedad JDBC "database name" o el método setDatabaseName() de la clase AS400JDBCDataSource para especificar la ASP a la que desea conectarse.

Todas las demás clases de Toolbox para Java (IFSFile, Print, DataQueues, etc.) utilizan la IASP especificada por la descripción de trabajo del perfil de usuario que se conecta al servidor.

Optimización de OS/400

El programa bajo licencia IBM Toolbox para Java está escrito en Java, por lo que se ejecuta en cualquier plataforma que tenga una máquina virtual Java (JVM) certificada. Las clases de IBM Toolbox para Java funcionan de manera parecida independientemente de dónde se ejecuten.

Junto con OS/400 se proporcionan clases que mejoran el comportamiento de IBM Toolbox para Java cuando se ejecuta en la máquina virtual Java de iSeries. El comportamiento de inicio de sesión y el rendimiento mejoran cuando la ejecución se realiza en la máquina virtual Java de iSeries y la conexión se establece con el mismo iSeries. Las clases adicionales forman parte de OS/400 a partir de la Versión 4 Release 3.

Habilitar la optimización

IBM Toolbox para Java se distribuye en dos paquetes: como programa bajo licencia independiente y con OS/400.

- Programa bajo licencia 5722-JC1. La versión de programa bajo licencia de IBM Toolbox para Java suministra archivos en el directorio siguiente:

  ```
  /QIBM/ProdData/http/public/jt400/lib
  ```

  Estos archivos no contienen optimizaciones de OS/400. Emplee estos archivos si desea obtener un comportamiento coherente con la ejecución de IBM Toolbox para Java en un cliente.
- OS/400. IBM Toolbox para Java también se proporciona con OS/400 en el directorio

  ```
  /QIBM/ProdData/OS400/jt400/lib
  ```

  Estos archivos sí contienen las clases que optimizan IBM Toolbox para Java al ejecutarse en la máquina virtual Java de iSeries.

Para obtener más información, consulte la nota 1 en la información acerca de los archivos JAR.

Consideraciones sobre el inicio de sesión

Con las clases adicionales proporcionadas con OS/400, los programas Java disponen de más opciones para suministrar información de nombre del sistema, ID de usuario y contraseña a IBM Toolbox para Java.

Al acceder a un recurso de iSeries, las clases de IBM Toolbox para Java deben tener un nombre del sistema, un ID de usuario y una contraseña.

- **Al ejecutarse en un cliente**, el programa Java es el que proporciona el nombre del sistema, el ID de usuario y la contraseña, o bien IBM Toolbox para Java recupera estos valores del usuario mediante un diálogo de inicio de sesión.
- **Al ejecutarse en la máquina virtual Java de iSeries**, IBM Toolbox para Java tiene una opción adicional. Puede enviar peticiones al servidor actual (local) utilizando el ID de usuario y la contraseña del trabajo que inició el programa Java.

Con las clases adicionales, también se puede utilizar el ID de usuario y la contraseña del trabajo actual cuando un programa Java que se ejecuta en un iSeries accede a los recursos existentes en otro iSeries. En este caso, el programa Java establece el nombre del sistema y luego utiliza el valor especial "*current" para el ID de usuario y la contraseña.

El programa Java solo puede establecer la contraseña en "*current" si se utiliza V4R4 o posterior para el acceso a nivel de registro. En caso contrario, cuando se utiliza el acceso a nivel de registro, "localhost" es válido para el nombre del sistema, y "*current" es válido para el ID de usuario; sin embargo, el programa Java debe suministrar la contraseña.

Un programa Java establece los valores de nombre del sistema, ID de usuario y contraseña en el objeto AS400.

Para utilizar el ID de usuario y la contraseña del trabajo, el programa Java puede usar "*current" como ID de usuario

y contraseña, o bien puede usar el constructor que no tiene los parámetros ID de usuario y contraseña.

Para utilizar el iSeries actual, el programa Java puede emplear "localhost" como nombre del sistema o bien emplear el constructor por omisión. Es decir:

```
AS400 system = new AS400();
```

equivale a

```
    AS400 system = new AS400("localhost", "*current", "*current");
```

En el siguiente ejemplo se crean dos objetos AS400. El comportamiento de los dos objetos es idéntico: los dos ejecutan un mandato en el iSeries actual utilizando el ID de usuario y la contraseña del trabajo. Uno de los objetos emplea el valor especial para el ID de usuario y la contraseña, mientras que el otro emplea el constructor por omisión y no establece el ID de usuario ni la contraseña.

```
        // Cree un objeto AS400. Debido a que se utiliza el
        // constructor por omisión y en ningún momento se establecen
        // el nombre del sistema, el ID de usuario ni la contraseña,
        // el objeto AS400 envía peticiones al iSeries local usando el
        // ID de usuario y la contraseña del trabajo. Si este programa
        // se ejecutase en un cliente, se pediría al usuario el
        // nombre del sistema, el ID de usuario y la contraseña.
    AS400 sys1 = new AS400();

        // Cree un objeto AS400. Este objeto envía
        // peticiones al iSeries local usando el ID de
        // usuario y la contraseña del trabajo. Este
        // objeto no funcionará en un cliente.
    AS400 sys2 = new AS400("localhost", "*current", "*current");

        // Cree dos objetos llamada a mandato que utilizan
        // los objetos AS400.
    CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
    CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

        // Ejecute los mandatos.
    cmd1.run();
    cmd2.run();
```

En el siguiente ejemplo, se crea un objeto AS400 que representa un segundo sistema iSeries. Como se utiliza el valor "*current", en el segundo iSeries (destino) se utiliza el ID de usuario y la contraseña del trabajo del iSeries que ejecuta el programa Java.

```
        // Cree un objeto AS400. Este objeto envía
        // peticiones a un segundo iSeries usando el ID de
        // usuario y la contraseña del trabajo del iSeries actual.
    AS400 sys = new AS400("mySystem.myCompany.com", "*current",
"*current");


        // Cree un objeto llamada a mandato que ejecute un
        // mandato en el iSeries destino.
    CommandCall cmd = new CommandCall(sys,"myCommand1");


        // Ejecute el mandato.
    cmd.run();
```

Mejoras en el rendimiento

Con las clases adicionales proporcionadas por OS/400, mejora el rendimiento de los programas Java que se ejecutan en la máquina virtual Java de iSeries. En algunos casos, el rendimiento mejora porque se utilizan menos funciones de comunicación y, en otros casos, porque se utiliza una API de iSeries, en vez de una llamada al programa servidor.

Tiempo de bajada más corto

Para bajar el número mínimo de archivos de clase de IBM Toolbox para Java, utilice el servidor proxy con la herramienta AS400ToolboxJarMaker.

Comunicación más rápida

Para todas las funciones de IBM Toolbox para Java salvo para JDBC y para el acceso al sistema de archivos integrado, los programas Java que se ejecutan en la máquina virtual Java de iSeries tendrán una ejecución más rápida. Ello se debe a que se utiliza menos código de comunicación cuando ésta se efectúa entre el programa Java y el programa servidor en el servidor que realiza la petición.

JDBC y el acceso al sistema de archivos integrado no se han optimizado porque ya existen servicios que agilizan la ejecución de estas funciones. En caso de ejecutarse en el iSeries, se puede utilizar el controlador JDBC de iSeries, en vez del controlador JDBC que se suministra junto con IBM Toolbox para Java. Para acceder a los archivos existentes en el servidor, puede utilizarse java.io en vez de las clases de acceso al sistema de archivos integrado que se suministran junto con IBM Toolbox para Java.

Llamada directa a las API de iSeries

La mejora en el rendimiento de las siguientes clases de IBM Toolbox para Java se debe a que estas clases llaman directamente a las API de iSeries, en vez de llamar a un programa servidor, para llevar a cabo la petición:

- Clases AS400Certificate
- CommandCall
- DataQueue
- ProgramCall
- Clases de acceso a base de datos a nivel de registro
- ServiceProgramCall
- UserSpace

La llamada directa a las API sólo se efectúa si el ID de usuario y la contraseña coinciden con el ID de usuario y la contraseña del trabajo que ejecuta el programa Java. Para obtener una mejora en el rendimiento, el ID de usuario y la contraseña deben coincidir con los del trabajo que inicia el programa Java. Para obtener unos resultados óptimos, utilice "localhost" para el nombre del sistema, "*current" para el ID de usuario y "*current" para la contraseña.

Cambios en la correlación de puertos

Se han realizado cambios en el sistema de correlación de puertos, de modo que se ha agilizado el acceso a un puerto. Antes de estos cambios, las peticiones de puerto se enviaban al correlacionador de puertos. Desde éste, el servidor iSeries determinaba qué puerto estaba disponible y devolvía dicho puerto al usuario para que lo aceptase. Ahora puede elegir entre indicar al servidor cuál es el puerto que se ha de utilizar o especificar que se tiene que usar el puerto por omisión. Esta opción evita que el servidor tenga que invertir tiempo en averiguar qué puerto se ha de emplear. Utilice el mandato WRKSRVTBLE para ver o cambiar la lista de puertos del servidor.

Para mejorar la correlación de puertos, se han añadido varios métodos a la clase AS400:

- getServicePort
- setServicePort
- setServicePortsToDefault

Cambios realizados en MRI

Ahora, los archivos de MRI se suministran dentro del programa IBM Toolbox para Java como archivos de clase, en vez de como archivos de propiedades. El servidor iSeries encuentra los mensajes más deprisa en los archivos de clase que en los archivos de propiedades. Ahora el método ResourceBundle.getString() se ejecuta más rápidamente porque los archivos de MRI se almacenan en el primer lugar en el que el sistema realiza la búsqueda. Otra ventaja de cambiar a archivos de clase es que el servidor puede localizar con mayor rapidez la versión traducida de una serie.

Conversores

Dos clases permiten una conversión más rápida y eficaz entre Java y el iSeries:

- Conversor binario: realiza una conversión entre las matrices de bytes Java y los tipos de datos simples Java.
- Conversor de tipo carácter: realiza una conversión entre los objetos de tipo serie Java y los paquetes de código de iSeries.

Además, ahora IBM Toolbox para Java incorpora sus propias tablas de conversión para más de 100 CCSID de uso habitual. Anteriormente, IBM Toolbox para Java remitía a Java casi toda la conversión de texto. Si Java no tenía la tabla de conversión correcta, IBM Toolbox para Java bajaba la tabla de conversión del servidor.

IBM Toolbox para Java lleva a cabo toda la conversión de texto para cualquier CCSID que conozca. Cuando encuentra un CCSID desconocido, intenta dejar que Java maneje la conversión. En ningún momento IBM Toolbox para Java intenta bajar una tabla de conversión del servidor. Este método reduce de forma notable el tiempo que tarda una aplicación de IBM Toolbox para Java en llevar a cabo la conversión de texto. No es necesaria ninguna acción del usuario para utilizar esta nueva conversión de texto; todas las mejoras en el rendimiento se producen en las tablas conversoras subyacentes.

Consejo para aumentar el rendimiento con el mandato Crear programa Java (CRTJVAPGM)

Si la aplicación Java se ejecuta en la máquina virtual Java (JVM) de iSeries, puede **aumentar el rendimiento de forma notable** si crea un programa Java desde un archivo zip o jar de IBM Toolbox para Java. Escriba el mandato **CRTJVAPGM** en una línea de mandatos de iSeries para crear el programa. (Encontrará más información en la ayuda en línea del mandato **CRTJVAPGM**.) Al utilizar el mandato **CRTJVAPGM**, se guarda el programa Java creado (que contiene las clases de IBM Toolbox para Java) cuando se inicia la aplicación Java. Al guardar el programa Java creado, podrá ahorrar tiempo de proceso de arranque. Se ahorra tiempo de proceso de arranque porque no es preciso que el programa Java se cree de nuevo en el servidor cada vez que se inicie la aplicación Java.

Si está utilizando la versión V4R2 o V4R3 de IBM Toolbox para Java, no puede ejecutar el mandato **CRTJVAPGM** con el archivo jt400.zip o jt400.jar ya que su tamaño es demasiado grande; pero sí puede ejecutarlo con el archivo jt400Access.zip. En la versión V4R3, el programa bajo licencia IBM Toolbox para Java incluye un archivo adicional, jt400Access.zip. El archivo jt400Access.zip solo contiene las clases de acceso, no las clases visuales.

Si ejecuta aplicaciones Java en un sistema con la versión V4R5 (o anterior), utilice jt400Access.zip. Si ejecuta aplicaciones Java en un sistema con la versión V5R1, utilice jt400Native.jar. El mandato **CRTJVAPGM** ya se ha ejecutado con el archivo jt400Native.jar.

Soporte de idioma nacional para Java

Java da soporte a un conjunto de idiomas nacionales, pero es un subconjunto de los idiomas soportados por el servidor.

Cuando se produce una discrepancia entre los idiomas (por ejemplo, cuando se trabaja en una estación de trabajo local que utiliza un idioma no soportado por Java) el programa bajo licencia IBM Toolbox para Java **puede emitir algunos mensajes de error en inglés**.

Servicio y soporte para IBM Toolbox para Java

Para obtener servicio y soporte, puede utilizar los recursos siguientes:

Información acerca de la resolución de problemas relacionados con Toolbox para Java
Utilice esta información a modo de ayuda para la resolución de los problemas que surjan al utilizar Toolbox para Java.

Foro de JTOpen/Toolbox para Java
Únase a la comunidad de programadores de Java que utilizan Toolbox para Java. Este foro es un método eficaz de obtener ayuda y recomendaciones de otros programadores de Java y, en ocasiones, de los propios desarrolladores de Toolbox para Java.

Soporte de servidor
Utilice el sitio Web de soporte de servidor de IBM para obtener información sobre las herramientas y los recursos que le ayudarán a perfeccionar la planificación y el soporte técnicos del servidor iSeries.

Soporte de software
Utilice el sitio Web de servicios de soporte de software de IBM para obtener información sobre la amplia gama de servicios de soporte de software que ofrece IBM.

Los servicios de soporte para IBM Toolbox para Java, 5722-JC1, se proporcionan bajo los términos y condiciones habituales en los productos de software de iSeries. Los servicios de soporte incluyen servicios de programa, soporte por conversación y servicios de consulta. Póngase en contacto con el representante local de IBM para obtener más información.

Los servicios de programa y el soporte por conversación permiten resolver defectos de programa de IBM Toolbox para Java, mientras que los servicios de consulta permiten resolver problemas de programación de aplicaciones y de depuración.

Los servicios de consulta admiten llamadas de interfaz de programas de aplicación (API) de IBM Toolbox para Java, a menos que:

- Se trate claramente de un defecto de la API de Java, defecto que debe demostrarse mediante su reproducción en un programa relativamente simple.
- Se trate de una consulta en que se solicitan aclaraciones de la documentación.
- Se trate de una consulta acerca de la ubicación de ejemplos o de documentación.

Los servicios de consulta admiten toda clase de ayuda para la programación, incluidos los ejemplos de programa proporcionados en el programa bajo licencia IBM Toolbox para Java. En Internet, en la página de presentación de iSeries puede encontrar ejemplos adicionales sin soporte.

Junto con el producto de programa bajo licencia IBM Toolbox para Java, se proporciona información de resolución de problemas. Si cree que hay un defecto potencial en la API de IBM Toolbox para Java, se necesitará un programa simple que demuestre el error.

Información relacionada acerca de IBM Toolbox para Java

La lista siguiente contiene sitios Web y temas de Information Center relacionados con la información de IBM Toolbox para Java.

Recursos de IBM Toolbox para Java

Consulte los siguientes sitios si desea adquirir más conocimientos sobre IBM Toolbox para Java:

- IBM Toolbox for Java and JTOpen: ofrece información sobre paquetes de servicio, consejos de rendimiento, ejemplos y mucho más. También puede bajar un paquete comprimido de esta información, incluidos los javadocs.
- IBM Toolbox for Java Frequently Asked Questions (FAQ): da respuesta a cuestiones relacionadas con el rendimiento, la resolución de problemas, JDBC y otros aspectos.
- IBM Toolbox for Java and JTOpen forum: proporciona un método eficaz para comunicarse con la comunidad de programadores de Java que utilizan Toolbox para Java y los propios desarrolladores de Toolbox para Java.

Recursos de IBM Toolbox para Java 2 Micro Edition

Consulte los siguientes sitios si desea adquirir más conocimientos sobre ToolboxME para iSeries y la implementación Java de las tecnologías inalámbricas:

- IBM Toolbox for Java and JTOpen: ofrece más información sobre ToolboxME para iSeries.
- IBM alphaWorks Wireless: facilita información sobre las nuevas tecnologías inalámbricas, con bajadas y enlaces a recursos de desarrollo.
- Sun Java 2 Platform, Micro Edition: proporciona información adicional sobre las tecnologías inalámbricas Java, tales como:
  - Máquina virtual K (KVM)
  - CLDC (Connected Limited Device Configuration)
  - MIDP (Mobile Information Device Profile)
- Java Wireless Developer: ofrece una amplia gama de información técnica para los desarrolladores de aplicaciones inalámbricas Java.
- Herramientas de desarrollo de aplicaciones inalámbricas:
  - IBM WebSphere Studio Device Developer
  - Java 2 Platform Micro Edition, Wireless Toolkit

Java

Java es un lenguaje de programación que permite desarrollar applets y aplicaciones orientadas a objetos portables. Consulte los siguientes sitios si desea adquirir más conocimientos sobre Java:

- IBM developerWorks Java technology zone: facilita información, formación y herramientas para ayudarle a utilizar Java, los productos de IBM y otras tecnologías a fin de crear soluciones para la empresa.

- IBM alphaWorks Java : contiene información sobre las nuevas tecnologías Java, con bajadas y enlaces a recursos de desarrollo.
- "The Source for Java Technology" from Sun Microsystems : ofrece información sobre los diversos usos de Java, incluidas las nuevas tecnologías.
- Java for iSeries, PartnerWorld for Developers : proporciona información sobre Java y sobre cómo pueden utilizar este lenguaje los socios comerciales de IBM y de qué manera lo utilizan en la vida real.

Java Naming and Directory Interface

- Java Naming and Directory Interface(TM) (JNDI) : ofrece una visión general de JNDI, información técnica, ejemplos y una lista de los proveedores de servicio disponibles.
- iSeries 400 Directory Services (LDAP) : facilita información sobre LDAP (Lightweight Directory Access Protocol) en OS/400.

Java Secure Socket Extension

- Java Secure Socket Extension (JSSE) : ofrece una breve visión general de JSSE y enlaces a más información.

Servlets

Los servlets son pequeños programas Java que se ejecutan en un servidor y actúan de mediadores entre las peticiones de uno o múltiples clientes (cada uno de los cuales se ejecuta en un navegador) y una o varias bases de datos. Como los servlets están programados en Java, pueden ejecutar peticiones como varias hebras en un solo proceso, lo que permite ahorrar recursos del sistema. Consulte los siguientes sitios si desea adquirir más conocimientos sobre los servlets:

- IBM Websphere, PartnerWorld for Developers : facilita información sobre el servidor de aplicaciones Web basado en servlets.
- Java Servlet technology : contiene información técnica, instrucciones y herramientas que le ayudarán a entender y utilizar los servlets.

XHTML

XHTML se considera como sucesor de HTML 4.0. Se basa en HTML 4.0, pero incorpora la extensibilidad de XML. Consulte los siguientes sitios si desea adquirir más conocimientos sobre XHTML:

- The Web Developer's Virtual Library : proporciona una introducción a XHTML, con ejemplos y enlaces a información adicional.
- W3C : proporciona información técnica sobre estándares XHTML y recomendaciones.

XML

XML (Extensible Markup Language) es un metalenguaje que permite describir y organizar la información de modo que sea fácilmente comprensible tanto para los humanos como para las máquinas. Un metalenguaje permite definir un lenguaje de códigos de documento y su estructura. Consulte los siguientes sitios si desea adquirir más conocimientos sobre XML:

- IBM developerWorks XML zone: ofrece un sitio dedicado al trabajo que realiza IBM con XML y a cómo permite facilitar el comercio electrónico.
- IBM alphaWorks XML: ofrece información sobre los nuevos estándares XML y herramientas relacionadas, con bajadas y enlaces a recursos de desarrollo.
- XML Support on iSeries, PartnerWorld for Developers: proporciona información sobre XML y sobre cómo pueden utilizar este lenguaje los socios comerciales de IBM y de qué manera lo utilizan en la vida real.
- W3C XML: ofrece recursos técnicos para los desarrolladores de XML.
- XML.com: facilita información actualizada sobre XML en la industria informática.
- XML.org: proporciona noticias e información sobre la comunidad de XML, con noticias del sector, calendarios de eventos y mucho más.
- XMLephant: ofrece recursos para aprender XML con enlaces a muchos otros sitios relacionados con XML.
- XML Cover Pages: ofrece un completo trabajo de referencia en línea para XML, SGML y estándares relacionados con XML, como XSL y XSLT.

Otras referencias

- IBM HTTP Server for iSeries: proporciona información, recursos y consejos sobre IBM HTTP Server para iSeries.
- iSeries Access for Windows: facilita información sobre iSeries Access para Windows, con bajadas, preguntas habituales y enlaces a sitios adicionales.
- IBM WebSphere Host On-Demand: contiene información sobre el emulador de navegador que ofrece soporte para S/390, iSeries y la emulación DEC/Unix.
- IBM Support and downloads: ofrece un portal al soporte de hardware y software de IBM.

Información de declaración de limitación de responsabilidad del código

Este documento contiene ejemplos de programación.

IBM le concede una licencia de copyright no exclusiva de uso de todos los ejemplos de código de programación a partir de los cuales puede generar funciones similares adaptadas a sus propias necesidades.

IBM proporciona todo el código de ejemplo sólo a efectos ilustrativos. Estos ejemplos no se han comprobado de forma exhaustiva en todas las condiciones. IBM, por lo tanto, no puede garantizar ni dar por sentada la fiabilidad, la utilidad ni el funcionamiento de estos programas.

Todos los programas contenidos aquí se proporcionan "TAL CUAL" sin garantías de ningún tipo. Las garantías implícitas de no incumplimiento, comerciabilidad y adecuación para un fin determinado se especifican explícitamente como declaraciones de limitación de responsabilidad.