System i

Программирование в IBM Toolbox for Java

Версия 6, выпуск 1

System i

Программирование в IBM Toolbox for Java

Версия 6, выпуск 1

Примечание

Перед началом работы с этой информацией и с описанным в ней продуктом ознакомьтесь со сведениями, приведенными в разделе “Примечания”, на стр. 771.

Данное издание относится к версии 6, выпуску 1, модификации 0 IBM Toolbox for Java (код продукта 5761-JC1), а также ко всем последующим выпускам и модификациям, если в новых изданиях не будет оговорено обратное. Данная версия работает не на всех моделях RISC и не работает на моделях CISC.

Содержание

IBM Toolbox for Java

IBM Toolbox for Java is a set of Java classes that allow you to use Java programs to access data on your system. You can use these classes to write client/server applications, applets, and servlets that work with data on your system. You can also run Java applications that use the IBM Toolbox for Java classes on the System i5 Java virtual machine (JVM).

R IBM Toolbox for Java uses the i5/OS Host Servers as access points to the system. Because IBM Toolbox for Java uses
L communication functions built into Java, you do not need to use IBM System i Access for Windows to use IBM
R Toolbox for Java. Каждый сервер связан с отдельным заданием сервера, которое обменивается данными по
R соединению с сокетом.

Примечание: Работая с примерами кода, вы подтверждаете свое согласие с условиями “Лицензия на исходный код и отказ от обязательств” на стр. 768.

What’s new for V6R1

This topic highlights the changes made to IBM Toolbox for Java in V6R1.

Enhancements to IBM Toolbox for Java JDBC support

Дополнительная информация о расширенных функциях JDBC приведена в разделе “Enhancements to IBM Toolbox for Java JDBC support for V6R1” на стр. 320. Дополнительная информация о новых свойствах JDBC приведена в разделе “IBM Toolbox for Java JDBC properties” на стр. 328.

Новые классы

The following classes have been added since the previous release of i5/OS. Все перечисленные классы входят в состав пакета ″com.ibm.as400.access″, если не оговорено противное.

- “AS400JDBCManagedConnectionPoolDataSource class” на стр. 68
- AS400JDBCManagedDataSource class
- “FileAttributes class” на стр. 48
- HistoryLog class
- “ObjectReferences class” на стр. 96
- UDFS class

Измененные классы

Существенно изменились следующие классы. Все перечисленные классы входят в состав пакета ″com.ibm.as400.access″, если не оговорено противное.

- Many of the JDBC classes
- CommandCall for Unicode enabled commands
- IFSFile
- ISeriesNetServer
- Permission classes
- ObjectList
- ObjectDescription
- User

Obtaining IBM Toolbox for Java

IBM Toolbox for Java is available in the following forms:

+ • The licensed program for IBM Toolbox for Java, 5761-JC1, Version 6 Release 1 (V6R1) installs on V5R4 and later
+ versions of i5/OS. From a client, IBM Toolbox for Java connects back to V5R3 and later versions of i5/OS.
- i5/OS also includes the non-graphical classes of IBM Toolbox for Java optimized for use when running IBM Toolbox for Java classes on the System i Java virtual machine (JVM). So if, for example, you do not have a need for the graphical functionality of the licensed program, you can still easily use IBM Toolbox for Java. За дополнительной информацией обратитесь к разделу Файлы Jar.
- IBM Toolbox for Java is also available in an open source version. Их и дополнительную информацию можно загрузить с Web-сайта JTOpen .

Совместимость

The IBM Toolbox for Java no longer ships the x4j400.jar (IBM XML parser). Рекомендуется применять в приложениях один из следующих анализаторов XML с поддержкой JAXP:

- Встроенный анализатор XML JDK версии 1.4 и выше
- Анализатор XML Apache Xerces, который можно загрузить с Web-сайта xml.apache.org
- One of the XML parsers that ship on i5/OS under /QIBM/ProdData/OS400/xml/lib

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

IBM Toolbox for Java no longer includes data400.jar. Классы из этого файла включены в файл jt400.jar. Удалите файл data400.jar из переменной CLASSPATH.

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

Если вы пользуетесь протоколом Secure Sockets Layer (SSL) для шифрования данных, которыми обмениваются клиент с сервером, потребуется Java Secure Socket Extension (JSSE).

To use all the IBM Toolbox for Java classes, use the Java 2 Platform, Standard Edition (J2SE). Для применения классов графического интерфейса или Graphical Toolbox необходим компонент Swing, входящий в состав J2SE. Для применения PDML необходима среда выполнения Java (JRE) версии 1.4 или выше.

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

Как узнать, что изменено

Для того чтобы упростить поиск изменений, в информации данной справочной системы применяются следующие обозначения:

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

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

PDF file for IBM Toolbox for Java

E You can view and print a PDF file of this information.

To view or download the PDF version of this document, select IBM Toolbox for Java (about 6895 KB).

You can view or download these related topic PDFs:

- IBM Developer Kit for Java (4585 KB)
- System i5 Debugger (134 KB)

Other information

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

Примечание: Файлы в пакете содержат ссылки на документы, которые не вошли в пакет, поэтому эти ссылки не работают.

Saving PDF files

Для сохранения файла в формате PDF на рабочей станции с целью последующего просмотра или печати выполните следующие действия:

E 1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Нажмите **Сохранить**.

Downloading Adobe Reader

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

Installing and managing IBM Toolbox for Java

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

Управление установкой IBM Toolbox for Java

Продукт IBM Toolbox for Java следует устанавливать только на тех клиентах, на которых он будет использоваться, или на общедоступной системе в сети. Your clients can be personal computers, dedicated workstations, or System i5 systems. It is important to remember that you can configure a System i5 or a partition of the server to be a client. В последнем случае IBM Toolbox for Java нужно установить в разделе сервера, относящемся к клиенту.

Предусмотрены различные способы установки IBM Toolbox for Java и управления этим продуктом:

- Индивидуальная установка, позволяющая установить и управлять отдельной копией IBM Toolbox for Java в каждой клиентской системе
- Сетевая установка на сервере, при которой продукт IBM Toolbox for Java устанавливается на общедоступном сетевом сервере

В следующих разделах приведено краткое сравнительное описание вышеуказанных методов с точки зрения эффективности и удобства. В дальнейшем способы разработки приложений на Java и управления ресурсами будут зависеть от выбранного метода (или сочетания методов) установки.

Индивидуальная установка

В некоторых случаях целесообразно устанавливать IBM Toolbox for Java отдельно в каждой клиентской системе. Основное преимущество отдельной установки IBM Toolbox for Java заключается в ускорении запуска приложений, использующих классы IBM Toolbox for Java, в клиентских системах.

Недостаток этого метода заключается в отсутствии автоматизации установки. Либо пользователь, либо специально созданная программа должны выполнять и контролировать установку нужной версии IBM Toolbox for Java на каждом клиенте по отдельности.

Сетевая установка на сервере

Можно также установить и обслуживать одну копию IBM Toolbox for Java на общедоступном сервере. Этот метод обладает следующими преимуществами:

- Все клиенты пользуются одной версией IBM Toolbox for Java
- Для обновления версии IBM Toolbox for Java для всех клиентов достаточно обновить один экземпляр продукта на сервере
- Обслуживание продукта на клиентах сводится к однократной корректировке переменной CLASSPATH

R Однако этот метод установки обладает существенным недостатком: увеличивается время запуска
R приложений IBM Toolbox for Java на клиентах. Кроме того, переменная CLASSPATH клиентов в этом случае
R будет ссылаться на сервер. You can use NetServer, which is integrated into i5/OS, or a different method that enables
L you to access files on the system, such as System i Access for Windows.

Установка IBM Toolbox for Java

Сначала следует выбрать метод установки IBM Toolbox for Java. В этих разделах описана установка IBM Toolbox for Java.

i5/OS requirements for IBM Toolbox for Java

This topic details the requirements that your environment must meet in order use IBM Toolbox for Java.

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

Необходимые компоненты i5/OS:

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

- Для запуска серверов хоста должен быть включен пользовательский профайл QUSER.
- Серверы хоста прослушивают сокеты и устанавливают соединения с клиентами. i5/OS Host Servers option is included with the base option of i5/OS. Дополнительные сведения приведены в разделе Управление серверами хоста.
- В i5/OS входит поддержка TCP/IP, позволяющая подключать сервер к сети. Дополнительные сведения приведены в разделе TCP/IP.

Starting required i5/OS options

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

1. Убедитесь в том, что включен пользовательский профайл QUSER.

2. Запустите серверы хоста i5/OS с помощью команды CL STRHOSTSVR. Введите **STRHOSTSVR *ALL** и нажмите **ENTER**.
3. Запустите сервер DDM TCP/IP с помощью команды STRTCPSVR. Введите **STRTCPSVR SERVER(*DDM)** и нажмите **ENTER**.

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

Many systems come with the IBM Toolbox for Java licensed product already installed. To see if IBM Toolbox for Java is already installed on your System i, complete the steps in this topic.

1. In System i Navigator, select and sign on to the system that you want to use.
2. В **дереве функций** (левая панель) откройте систему, затем **Настройка и обслуживание**.
3. Откройте **Программное обеспечение**, затем **Установленные продукты**.
4. In the **Details** pane (the right pane), look in the **Product** column for 5761jc1. Если оно есть, продукт IBM Toolbox for Java установлен на данном сервере.

   Примечание: Кроме того, определить, установлен ли продукт IBM Toolbox for Java, можно с помощью опции 11 команды CL Перейти к меню (**GO MENU(LICPGM)**).

Если лицензионный продукт IBM Toolbox for Java не установлен, IBM Toolbox for Java можно установить.

If a previous version of IBM Toolbox for Java is installed, first delete the currently installed version then install the IBM Toolbox for Java licensed product. Для того чтобы предотвратить возможные неполадки, перед удалением текущей версии IBM Toolbox for Java рекомендуется создать ее резервную копию.

Проверка состояния профайла QUSER:

The i5/OS Host Servers start under the QUSER user profile, so you first need to ensure that the QUSER profile is enabled in order to run IBM Toolbox for Java in a client/server environment.

Проверка состояния профайла QUSER

Для того чтобы узнать состояние профайла QUSER, выполните следующие действия:

1. On a command line, type `DSPUSRPRF USRPRF(QUSER)` and press **Enter**.
2. Убедитесь, что в поле **Состояние** указано значение *ENABLED. Если в поле указано другое состояние, измените профайл QUSER.

Изменение пользовательского профайла QUSER:

If the QUSER profile is not *ENABLED, you must enable it to start the i5/OS Host Servers. Кроме того, пароль профайла QUSER должен быть отличен от *NONE. Если это не так, сбросьте пароль.

Для разблокирования профайла QUSER из командной строки выполните следующие действия:

1. Введите `CHGUSRPRF USRPRF(QUSER)` и нажмите **ENTER**.
2. Измените поле **Состояние** на *ENABLED и нажмите **ENTER**.

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

Зависимость от других лицензионных программ:

Для применения некоторых функций IBM Toolbox for Java требуется установить дополнительные лицензионные программы.

Программа просмотра буферных файлов

Если вы собираетесь использовать функцию просмотра буферных файлов IBM Toolbox for Java (класс SpooledFileViewer), то убедитесь, что на вашем сервере установлен компонент хоста 8 (AFP Compatibility Fonts).

Примечание: Классы SpooledFileViewer, PrintObjectPageInputStream и PrintObjectTransformedInputStream можно применять только при подключении к операционной системе выпуска V4R4 или выше.

SSL

Если вы планируете использовать SSL, необходимо установить следующие продукты:

R • IBM HTTP Server for System i licensed program, 5761-DG1
- Компонент 34 i5/OS (Диспетчер цифровых сертификатов)

Дополнительная информация об SSL приведена в разделе “Сравнение Secure Sockets Layer и Java Secure Socket Extension” на стр. 352.

Для работы с аплетами, сервлетами или SSL требуется установить сервер HTTP.

R If you want to use applets, servlets, or SSL on the System i platform, you must set up an HTTP server and install the
R class files on the system. For more information about the IBM HTTP Server, see the IBM HTTP Server Webmaster’s
R Guide, GC41-5434, at the following URL: http://www.ibm.com/eserver/iseries/products/http/docs/doc.htm .
R Руководство Web-мастера представлено в форматах HTML и PDF.

Дополнительная информация о Диспетчере цифровых сертификатов и создании и применении сертификатов с помощью IBM HTTP Server приведена в разделе Управление цифровыми сертификатами.

Compatibility with different levels of i5/OS:

Так как продукт IBM Toolbox for Java может применяться как на сервере, так и на клиенте, то вопросы совместимости влияют как на работу сервера, так и на установление соединения клиента с сервером.

Работа с IBM Toolbox for Java на серверах

To install IBM Toolbox for Java (licensed program 5761-JC1 V6R1), the server must be running one of the following:

+ • i5/OS Version 6 Release 1
- i5/OS Version 5 Release 4

В системе можно установить только одну версию лицензионной программы IBM Toolbox for Java Для установки другой версии необходимо сначала удалить установленную версию лицензионной программы IBM Toolbox for Java.

Применение IBM Toolbox for Java для установления соединения клиента с сервером.

R На клиенте и сервере, между которыми устанавливается соединение, могут быть установлены различные
R версии IBM Toolbox for Java. To use IBM Toolbox for Java to access data and resources on a System i, the **server to**
R **which you are connecting** must be running one of the following:

+ • i5/OS Version 6 Release 1
- i5/OS Version 5 Release 4
- i5/OS Version 5 Release 3

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

Примечание: IBM Toolbox for Java не поддерживает совместимость со следующими версиями. You cannot install IBM Toolbox for Java on or use it to connect to a server that runs a more recent version of i5/OS. For example, if you are using the version of IBM Toolbox for Java that ships with i5/OS V5R2, you cannot install it on or connect to a server that runs i5/OS V5R4.

Программы	Ships with i5/OS	Installs on i5/OS	Connects back to i5/OS
5722-JC1 V5R3M0	V5R3	V5R1 и выше	V5R1 и выше
5722-JC1 V5R4M0	V5R4	V5R3 и более новые	V5R2 и выше
+ 5761-JC1 V6R1	V6R1	V5R4 and later	V5R3 и более новые

Native optimizations when running on the i5/OS JVM:

Для того чтобы работа классов IBM Toolbox for Java отвечала требованиям пользователей i5/OS, предусмотрен набор функций внутренней оптимизации. The optimizations affect operation of IBM Toolbox for Java only when running on the i5/OS JVM.

The native optimizations version of IBM Toolbox for Java ships with i5/OS on your system. Внутренняя оптимизация:

- Вход в систему: Если в объекте AS400 не указан ИД пользователя или пароль, применяется ИД пользователя и пароль текущего задания
- Вызов API i5/OS напрямую без вызовов серверов хоста с помощью сокетов:
  - Доступ к базам данных на уровне записей, доступ к очередям данных и пользовательским пространствам при выполнении требований к защите.
  - Вызов программ и команд при выполнении требований к защите и обеспечении поддержки нитей.

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

Для применения оптимизации не нужно вносить изменения в приложения на Java. IBM Toolbox for Java автоматически использует оптимизацию, если это возможно.

Для повышения производительности следует использовать файл jar, содержащий функции внутренней оптимизации i5/OS. Дополнительные сведения приведены в примечании 1 к разделу Файлы Jar.

+ When you do not use the jar file that includes i5/OS native optimizations, IBM Toolbox for Java works as if it is
+ running on a client.

ToolboxME requirements:

Your workstation, wireless device, and server must meet certain requirements (listed below) for developing and running IBM Toolbox for Java 2 Micro Edition applications.

Although Toolbox ME is considered a part of IBM Toolbox for Java, it is not included in the licensed product. ToolboxME (jt400Micro.jar) is included in the open source version of Toolbox for Java, called JTOpen. You must separately download and set up ToolboxME, which is contained in JTOpen.

Требования

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

Требования к рабочей станции

Workstation requirements for developing ToolboxME applications:

+ • A supported version of Java Standard Edition
  • Java virtual machine for wireless devices
  • Симулятор или эмулятор беспроводного устройства

Требования к беспроводному устройству

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

Требования к серверу

Server requirements for using ToolboxME applications:

- Класс MEServer, который входит в IBM Toolbox for Java или в последнюю версию JTOpen
- Требования к i5/OS для работы с IBM Toolbox for Java

Требования к рабочей станции IBM Toolbox for Java

Убедитесь, что рабочая станция отвечает следующим требованиям.

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

Требования к рабочей станции для запуска приложений IBM Toolbox for Java:

Для создания и запуска приложений IBM Toolbox for Java рабочая станция должна отвечать следующим требованиям.

- Рекомендуется применять поддерживаемую виртуальную машину Java: Java 2 Standard Edition (J2SE). Для применения многих новых функций IBM Toolbox for Java требуется версия JVM 1.4 или выше.
- Для применения классов графического интерфейса или Graphical Toolbox необходим компонент Swing, входящий в состав J2SE. Swing версии 1.1 можно также загрузить с Web-сайта Sun Java Foundation Classes  . Протестированы следующие среды:
  - Windows 2000
  - Windows XP
  - AIX версии 4.3.3.1
  - Sun Solaris версии 5.7
  - i5/OS версии 5 выпуска 3 или более поздней версии
  - Linux (Red Hat 7.0)
- Стек TCP/IP.

Требования к рабочей станции для выполнения аплетов IBM Toolbox for Java:

Для создания и запуска приложений IBM Toolbox for Java рабочая станция должна отвечать следующим требованиям.

- Браузер, совместимый с виртуальной машиной Java. Протестированы следующие среды:
  - Netscape Communicator 4.7, using a current Java plug-in

    Примечание: Продукт IBM Toolbox for Java теперь нельзя запустить в стандартной JVM браузеров Netscape Navigator и Microsoft Internet Explorer. Для того чтобы запустить в браузере аплет с классами IBM Toolbox for Java, нужно установить специальный встраиваемый модуль, например, Sun Java 2 Runtime Environment (JRE)  .
- Установленный и настроенный стек TCP/IP.
+ • The workstation must connect to a server that is running i5/OS V5R3 or later

ToolboxME requirements:

Your workstation, wireless device, and server must meet certain requirements (listed below) for developing and running IBM Toolbox for Java 2 Micro Edition applications.

Although Toolbox ME is considered a part of IBM Toolbox for Java, it is not included in the licensed product. ToolboxME (jt400Micro.jar) is included in the open source version of Toolbox for Java, called JTOpen. You must separately download and set up ToolboxME, which is contained in JTOpen.

Требования

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

Требования к рабочей станции

Workstation requirements for developing ToolboxME applications:

+ • A supported version of Java Standard Edition
  • Java virtual machine for wireless devices
  • Симулятор или эмулятор беспроводного устройства

Требования к беспроводному устройству

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

Требования к серверу

Server requirements for using ToolboxME applications:

- Класс MEServer, который входит в IBM Toolbox for Java или в последнюю версию JTOpen
- Требования к i5/OS для работы с IBM Toolbox for Java

Требования к рабочей станции Swing IBM Toolbox for Java:

Продукт IBM Toolbox for Java поддерживает Swing 1.1 в OS/400, начиная с версии V4R5. Переход на Swing требовал внесения изменений в классы IBM Toolbox for Java. Поэтому, если ваши программы применяют Graphical Toolbox или классы vaccess из выпусков до V4R5, необходимо также внести изменения в программы.

Кроме этого, при запуске программ необходимо, чтобы классы Swing находились в каталоге, описанном в CLASSPATH. Классы Swing являются частью Java 2 Platform. Если у вас нет пакета Java 2 Platform, можно загрузить классы Swing 1.1 с сервера Sun Microsystems, Inc. 

Installing IBM Toolbox for Java on your system

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

Примечание: Оригинальная версия IBM Toolbox for Java входит в комплект поставки i5/OS. So, if you want to use IBM Toolbox for Java only on your server, you do not have to install the licensed product. Дополнительные сведения о версии IBM Toolbox for Java, поставляемой вместе с системой, приведены в разделе Файлы Jar: примечание 1.

Перед установкой IBM IBM Toolbox for Java нужно убедиться, что применяемая версия i5/OS отвечает требованиям для применения IBM Toolbox for Java. Некоторые серверы поставляются с заранее установленным продуктом IBM Toolbox for Java. You may want to determine whether the IBM Toolbox for Java licensed product is already installed on your server.

Установка IBM Toolbox for Java

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

L **Using System i Navigator to install IBM Toolbox for Java**

L To install IBM Toolbox for Java using System i Navigator, complete the following steps:

L 1. In System i Navigator, sign on to the system that you want to use.
2. Откройте **Мои соединения** в левой панели (Дерево функций).
3. В разделе **Мои соединения** щелкните правой кнопкой мыши на системе, в которой нужно установить IBM Toolbox for Java.
4. Выберите пункт **Выполнить команду**.
5. В окне **Восстановить лицензионную программу (RSTLICPGM)** введите следующие сведения, а затем нажмите кнопку **OK**:
   - Product: 5761JC1
   - Устройство: имя нужного устройства или файла сохранения

   Примечание: Для просмотра дополнительной информации щелкните на значке **Справка** в окне диалога **Восстановление лицензионной программы (RSTLICPGM)**.

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

1. Откройте **Централизованное управление**.
2. Откройте **Текущие задачи**.
3. В дереве **Текущие задачи** выберите **Команды**.
4. Щелкните на нужной задаче **Выполнение команды** в панели Сведения.

Using the command line to install IBM Toolbox for Java

R To install IBM Toolbox for Java from a command line, complete the following steps:

R 1. On a command line, use the CL Go to Menu command. Введите **GO MENU(LICPGM)** и нажмите **ENTER**.
2. Выберите опцию **11. Установить лицензионную программу**.
3. Select **5761-JC1 IBM Toolbox for Java**.

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

Установка IBM Toolbox for Java на рабочей станции

Перед установкой IBM Toolbox for Java убедитесь, что выполнены требования к рабочей станции.

Оптимальный способ установки IBM Toolbox for Java на рабочей станции зависит от того, как вы планируете управлять установкой:

- Для установки IBM Toolbox for Java на отдельных клиентах нужно скопировать файлы JAR на рабочую станцию и соответствующим образом изменить значение переменной CLASSPATH.
- Для применения экземпляра IBM Toolbox for Java, установленного на сервере, достаточно указать в переменной CLASSPATH путь к этому экземпляру на сервере. To point your workstation CLASSPATH to the server, your server must have i5/OS NetServer installed.

В этом разделе приведены инструкции по копированию файлов с классами на рабочую станцию. For more information about setting the CLASSPATH on your workstation, refer to the operating system documentation for your workstation or information available at the Sun Java Web site .

Примечание: Для применения классов IBM Toolbox for Java при работе с приложением система должна соответствовать требованиям к i5/OS.

Файлы классов IBM Toolbox for Java хранятся в нескольких файлах jar, поэтому эти файлы необходимо скопировать на рабочую станцию. Сведения о том, какие файлы .jar нужны для выполнения отдельных функций IBM Toolbox for Java, приведены в разделе Файлы Jar.

Пример: Копирование файла jt400.jar

Предположим, что вам нужно скопировать файл jt400.jar (в этом файле хранятся основные классы IBM Toolbox for Java).

Для копирования файла jar вручную выполните следующие действия:

1. Найдите файл jt400.jar в следующем каталоге: /QIBM/ProdData/HTTP/Public/jt400/lib
2. Скопируйте файл jt400.jar с сервера на рабочую станцию. Это можно сделать разными способами:
   - Use System i Access for Windows to map a network drive on your workstation to the server, then copy the file.
   - Передать файл на рабочую станцию по FTP (в двоичном режиме).
3. Обновить переменную среды CLASSPATH на рабочей станции.
   - Например, если вы работаете в Windows NT и скопировали файл jt400.jar в C:\jt400\lib, добавьте следующую строку в конце CLASSPATH:

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

Кроме того, можно установить версию IBM Toolbox for Java с открытым исходным кодом, которая называется JTOpen. С дополнительной информацией по JTOpen можно ознакомиться на Web-сайте IBM Toolbox for Java и JTOpen .

Файлы Jar:

The IBM Toolbox for Java is shipped as a set of jar files. В каждом файле хранятся пакеты Java, предоставляющие определенные функции. Для экономии дискового пространства можно установить только те файлы jar, которые содержат необходимые вам функции.

Для применения файла jar необходимо добавить имя каталога, в котором он расположен, в переменную среды CLASSPATH.

В приведенной ниже таблице показано, какие файлы jar необходимо добавить в переменную CLASSPATH для работы с файлами из указанного пакета.

IBM Toolbox for Java package or function	**Файлы Jar, которые нужно добавить в переменную CLASSPATH**
Классы доступа	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1, либо jt400Proxy.jar в среде с серверами proxy
“Класс CommandHelpRetriever” на стр. 248	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1 и анализатор XML и обработчик XSLT Примечание 2
Класс CommandPrompter Примечание 3	jt400.jar, jui400.jar, util400.jar Примечание 4 и анализатор XML Примечание 2
Классы трассировки соединений	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1

IBM Toolbox for Java package or function	Файлы Jar, которые нужно добавить в переменную CLASSPATH
Классы HTML	jt400.jar Примечание 1 и jt400Servlet.jar (клиентский), либо jt400Native.jar (серверный) Примечание 1
Класс HTMLDocument	Те же файлы jar, которые необходимы для работы с классами HTML, а также анализатор XML и обработчик XSLT Примечание 2
Классы JCA	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1
GUI источника данных JDBC	jt400.jar (клиентский) Примечание 1 и jui400.jarПримечание 5
Сообщения системы и сообщения об ошибках NLS	jt400Mri_lang_cntry.jar Примечание 6
PCML (разработки и выполнения, проанализированный) Примечание 7	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1, Примечание 8, анализатор XML Примечание 2
PCML (времени выполнения, двоичный)	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1, Примечание 8
PDML (разработка) Примечание 3	uitools.jar, jui400.jar, util400.jar Примечание 4 и анализатор XML Примечание 2
PDML (выполнения, проанализированный) Примечание 3	jui400.jar, util400.jar Примечание 4 и анализатор XML Примечание 2
PDML (выполнения, двоичный) Примечание 3	jui400.jar и util400.jar Примечание 4
Классы составителя отчетов	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1 файлы jar программы создания отчетов Примечание 9, а также анализатор XML и обработчик XSLT Примечание 2
Классы ресурсов	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1
RFML	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1, а также анализатор XML Примечание 2
Классы защиты	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1, либо jt400Proxy.jar в среде с серверами proxy
Классы сервлетов	jt400.jar Примечание 1 и jt400Servlet.jar (клиентский), либо jt400Native.jar (серверный) Примечание 1
System i5 Debugger Note 3	jt400.jar (клиентский) Примечание 1 и tes.jar
“IBM Toolbox for Java 2 Micro Edition” на стр. 353	jt400Micro.jar (клиентский) Примечание 10 и jt400.jar (серверный), либо jt400Native.jar (серверный)Примечание 1
Классы Vaccess	jt400.jar (клиентский) Примечание 1
XPCML	jt400.jar (клиентский) или jt400Native.jar (серверный) Примечание 1, а также анализатор XML и обработчик XSLT Примечание 2

+ **Note 1:** Do not put both jt400.jar and jt400Native.jar in your CLASSPATH. Choose the .jar file most appropriate for
+ your environment and use only that .jar in your CLASSPATH.

Note 2: Some of the IBM Toolbox for Java classes are in more than one jar file:

- **jt400.jar** - классы доступа, трассировки соединений, JCA, поддержки JDBC, MEServer, PCML, ресурсов, RFML, защиты, утилит, графического интерфейса и XPCML.
- **jt400.zip** - Используйте файл jt400.jar вместо jt400.zip. jt400.zip is shipped to retain compatibility with previous releases of IBM Toolbox for Java.
- **jt400Access.zip** - Те же классы, что и в jt400.jar, за исключением классов графического интерфейса. jtAccess400.zip is shipped to retain compatibility with previous releases of IBM Toolbox for Java. Используйте файл jt400.jar вместо jt400.zip.

- **jt400Native.jar** - классы доступа, HTML, MEServer, PCML, ресурсов, RFML, защиты, XPCML и внутренней оптимизации. Native optimizations is a set of classes (fewer than 20) that take advantage of System i5 function when running on the i5/OS JVM. Because jt400Native.jar contains the native optimizations, when running on thei5/OS JVM, use jt400Native.jar instead of jt400.jar. jt400Native.jar ships with i5/OS and resides in directory /QIBM/ProdData/OS400/jt400/lib.
- **jt400Native11x.jar** - Используйте jt400Native.jar вместо jt400Native11x.jar. jt400Native11x.jar поставляется для совместимости с предыдущими выпусками IBM Toolbox for Java.

Note 3: When you must use an XML parser or XSLT processor, make sure that they are JAXP-compliant. Дополнительная информация приведена на следующей странице:

"Анализатор XML и обработчик XSLT" на стр. 418

Note 4: Using CommandPrompter, PDML, or the System i5 Debugger also requires one additional jar file that is not part of IBM Toolbox for Java: jhall.jar. For more information about downloading jhall.jar, see the Sun JavaHelp Web site .

Note 5: util400.jar contains System i-specific classes for formatting input and for using the command line prompter. Этот файл необходим для применения класса CommandPrompter. Для применения PDML файл util400.jar не требуется, однако он содержит некоторые полезные функции.

Note 6: jui400.jar contains the classes necessary to use the JDBC DataSource GUI interface. Файл jt400.jar (Примечание 1) содержит классы, необходимые для применения всех остальных функций JDBC.

Note 7: jt400Mri_xx_yy.jar contains translated messages, including strings contained in exception messages, dialogs, and output from other normal processing. В файле jt400Mri_lang_cntry.jar, lang = Код языка ISO, a cntry = Код страны или региона ISO, применяемый для перевода содержимого. В некоторых случаях Код страны или региона ISO не применяется. Installing a particular national language version of the IBM Toolbox for Java licensed program on the system installs the appropriate jt400Mri_lang_cntry.jar file. If the language is not supported, the install defaults to the English version, which is included in the IBM Toolbox for Java jar files.

- For example, installing the German language version of licensed program 5761-JC1 installs the German language jar file, jt400Mri_de.jar.

Для поддержки нескольких языков укажите в переменной CLASSPATH несколько файлов jar. Java будет применять сообщения из того файла, который соответствует текущей локали.

Note 8: Serializing your PCML file during development has two benefits:

1. Файл PCML анализируется только во время разработки, а не во время выполнения
2. Для запуска приложения пользователям нужно добавлять в переменную CLASSPATH меньше файлов jar

Для анализа файла PCML во время разработки необходимы классы PDML времени выполнения из файла data.jar либо jt400.jar, а также программа анализа PCML из файла x4j400.jar. Для запуска приложения, преобразованного в последовательность байт, требуется только файл jt400.jar. For more information, see "Building System i5 program calls with PCML" на стр. 382.

Note 9: Use jt400.jar and jt400Native.jar instead of data400.jar. Файл data400.jar содержит классы выполнения PCML, которые также входят в jt400.jar и jt400Native.jar (Примечание 1). data400.jar is shipped to retain compatibility with previous releases of IBM Toolbox for Java.

Note 10: Copies of the ReportWriter classes are in more than one jar file:

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

R If your application streams PCL data to an i5/OS spooled file, you must make the access classes available by using the
R appropriate jar file (Note 1). Для записи данных PCL в буферный файл необходимы классы AS400,
R OutputQueue, PrintParameterList и SpooledFileOutputStream. Дополнительная информация приведена в разделе
R Классы ReportWriter.

Note 11: jt400Micro.jar does not contain the classes needed to run MEServer, which reside in both jt400.jar and jt400Native.jar (Note 1). jt400Micro.jar is available only from the IBM Toolbox for Java and JTOpen Web site .

Свойства системы

Системные свойства позволяют настроить различные параметры IBM Toolbox for Java.

Например, системные свойства позволяют задать сервер Proxy или уровень трассировки. Кроме того, они позволяют задать параметры программы во время ее работы, не требуя перекомпиляции кода. Системные свойства аналогичны переменным среды - их изменение во время выполнения программы не отражается на работе до следующего запуска.

Системные свойства можно задать несколькими способами:

- **С помощью метода java.lang.System.setProperties()**

  В программе системные свойства можно задать с помощью метода java.lang.System.setProperties().

  Например, в приведенном ниже фрагменте кода свойству `com.ibm.as400.access.AS400.proxyServer` присваивается значение `hqoffice`:

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

- **С помощью опции -D команды java**

  Во многих средах системные свойства можно задать при запуске приложения из командной строки с помощью опции `-D` команды `java`.

  Например, следующая программа запускает приложение Inventory со свойством `com.ibm.as400.access.AS400.proxyServer`, равным `hqoffice`:

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

- **С помощью файла jt400.properties**

  В некоторых случаях неэффективно задавать системные свойства при каждом запуске приложения. Вместо этого системные свойства IBM Toolbox for Java могут быть заданы в файле jt400.properties, который просматривается при запуске приложения, как если бы он входил в состав пакета com.ibm.as400.access. Для работы с файлом jt400.properties поместите его в каталог com/ibm/as400/access, указанный в переменной CLASSPATH.

  Например, для того чтобы присвоить свойству `com.ibm.as400.access.AS400.proxyServer` значение `hqoffice`, добавьте в файл jt400.properties следующую строку:

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

  В файлах свойств обратная косая черта (\) играет роль Escape-символа. Для ввода обратной косой черты укажите этот символ дважды (\\).

  Для изменения значений свойств отредактируйте данный пример.

- **С помощью класса Properties**

  В некоторых браузерах для загрузки файлов свойств требуется изменить параметры защиты. Однако большинство браузеров разрешают указывать свойства в файлах .class, поэтому свойства IBM Toolbox for Java можно задать с помощью класса com.ibm.as400.access.Properties, расширяющего класс java.util.Properties.

  Ниже приведен фрагмент кода на Java, в котором свойству `com.ibm.as400.access.AS400.proxyServer` присваивается значение `hqoffice`:

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

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

Для настройки файла свойств отредактируйте пример исходного файла Properties.java.

Если системное свойство IBM Toolbox for Java задано несколькими из описанных выше способов, то применяется значение свойства с максимальным приоритетом. Следующий список упорядочен по убыванию приоритета:

1. Системное свойство, заданное в программе с помощью метода `java.lang.System.setProperties()`
2. Системное свойство, заданное с помощью опции `-D` команды `java`
3. Системное свойство, заданное в классе Properties
4. Системное свойство, заданное в файле jt400.properties

IBM Toolbox for Java поддерживает следующие системные свойства:

- “Свойства сервера Proxy”
- “Свойства трассировки” на стр. 16
- “Свойства CommandCall/ProgramCall ” на стр. 16
- “Свойства FTP ” на стр. 16
- “Свойства соединения ” на стр. 16

Свойства сервера Proxy

Свойство сервера Proxy	**Описание**
com.ibm.as400.access.AS400.proxyServer	Задает имя хоста и порт сервера Proxy в следующем формате: `имя-хоста:номер-порта` Номер порта необязателен.
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval	Задает периодичность, с которой сервер Proxy выполняет процедуру поиска простаивающих соединений (в секундах). Сервер Proxy запускает отдельную нить для поиска клиентов, не обменивающихся данными с сервером. Данное свойство позволяет задать частоту выполнения этой нити.
com.ibm.as400.access.TunnelProxyServer.clientLifetime	Задает время простоя клиента (в секундах), по истечении которого сервер Proxy удаляет ссылки на объекты, для того чтобы они были удалены программой сборки мусора JVM. Сервер Proxy запускает отдельную нить для поиска клиентов, не обменивающихся данными с сервером. Данное свойство позволяет задать время простоя клиента перед выполнением сбора мусора.

Свойства трассировки

Свойство трассировки	Описание
com.ibm.as400.access.Trace.category	Задает применяемые категории трассировки. В качестве значения можно указать список категорий трассировки через запятую. The complete list of trace categories is defined in the Trace class.
com.ibm.as400.access.Trace.file	Задает файл вывода трассировки. По умолчанию применяется файл System.out.
com.ibm.as400.access.ServerTrace.JDBC	Задает категории трассировки задания сервера JDBC. Информация о поддерживаемых значениях приведена в разделе Свойство трассировки сервера JDBC.

Свойства CommandCall/ProgramCall

Свойство CommandCall/ProgramCall	Описание
com.ibm.as400.access.CommandCall.threadSafe	Указывает, поддерживают ли объекты CommandCall нити. Если указано значение `true`, все объекты CommandCall считаются поддерживающими нити. Если указано значение `false`, все объекты CommandCall считаются не поддерживающими нити. Это свойство игнорируется, если для объекта CommandCall был вызван метод `CommandCall.setThreadSafe(true/false)` или `AS400.setMustUseSockets(true)`.
com.ibm.as400.access.ProgramCall.threadSafe	Указывает, поддерживают ли объекты CommandCall нити. Если указано значение `true`, все объекты ProgramCall считаются поддерживающими нити. Если указано значение `false`, все объекты ProgramCall считаются не поддерживающими нити. Это свойство игнорируется, если для объекта ProgramCall был вызван метод `ProgramCall.setThreadSafe(true/false)` или `AS400.setMustUseSockets(true)`.

Свойства FTP

Свойство FTP	Описание
com.ibm.as400.access.FTP.reuseSocket	Указывает, будет ли сокет многократно использоваться для нескольких операций передачи файлов (с помощью одного экземпляра FTP) в ″активном″ режиме. Если это свойство равно `true`, то сокет используется многократно. Если свойство равно `false`, то для каждой передачи создается новый сокет. Если для объекта FTP был вызван метод `FTP.setReuseSocket(true/false)`, то это свойство игнорируется.

Свойства соединения

Свойство соединения	Описание
com.ibm.as400.access.AS400.signonHandler	Задает обработчик по умолчанию для входа в систему. Если для объекта AS400 был выл вызван метод `AS400.setSignonHandler()` или `AS400.setDefaultSignonHandler()`, то это свойство игнорируется.

Пример: Файл свойств

This example shows properties for the proxy server, trace categories, command calls, program calls, file transfers, and connections.

```
#=========================================================#
# IBM Toolbox for Java                                    #
#---------------------------------------------------------#
# Пример файла свойств                                    #
#                                                         #
# Этот файл с именем jt400.properties должен находиться   #
# в каталоге com/ibm/as400/access, указанном в переменной #
# classpath.                                              #
#=========================================================#

#---------------------------------------------------------#
# Системные свойства сервера Proxy                        #
#---------------------------------------------------------#

# Данное системное свойство задает имя хоста и номер порта
# сервера Proxy в формате имя-хоста:номер-порта
# Номер порта необязателен.
com.ibm.as400.access.AS400.proxyServer=hqoffice
# Данное системное свойство задает периодичность выполнения
# the proxy server will look for idle connections. # Сервер Proxy запускает отдельную нить для поиска клиентов,
# не обменивающихся данными с сервером. Данное свойство
# позволяет задать частоту выполнения этой нити.
com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval=7200

# Данное системное свойство задает время (в секундах)
# простоя клиента перед удалением соединения. Сервер
# Proxy запускает отдельную нить для поиска клиентов,
# не обменивающихся данными с сервером. Данное свойство
# позволяет задать время простоя соединения перед удалением.
com.ibm.as400.access.TunnelProxyServer.clientLifetime=2700


#---------------------------------------------------------#
# Свойства трассировки                                    #
#---------------------------------------------------------#

# Данное системное свойство задает применяемые категории трассировки.
# В качестве значения можно указать список категорий через запятую.
# Полный список категорий трассировки задан в классе
# Trace.
com.ibm.as400.access.Trace.category=error,warning,information

# Данное системное свойство задает файл вывода трассировки.
# По умолчанию применяется файл System.out.
com.ibm.as400.access.Trace.file=c:\\temp\\trace.out


#---------------------------------------------------------#
# Свойства вызова команд                                  #
#---------------------------------------------------------#

# Это системное свойство указывает, поддерживают ли объекты
# CommandCall нити. Если указано значение true - все объекты
# CommandCall считаются поддерживающими нити. Если указано false -
# все объекты считаются не поддерживающими нити. Это свойство
# игнорируется, если для объекта CommandCall был вызван метод
# CommandCall.setThreadSafe(true/false) или
# AS400.setMustUseSockets(true).
com.ibm.as400.access.CommandCall.threadSafe=true


#---------------------------------------------------------#
# Свойства вызова программ                                #
```

```
#---------------------------------------------------------#

# Это системное свойство указывает, поддерживают ли объекты
# ProgramCall нити. Если указано true - все объекты ProgramCall
# считаются поддерживающими нити. Если указано значение false -
# все объекты считаются не поддерживающими нити. Это свойство
# игнорируется, если для объекта ProgramCall был вызван метод
# ProgramCall.setThreadSafe(true/false) или
# AS400.setMustUseSockets(true).
com.ibm.as400.access.ProgramCall.threadSafe=true


#---------------------------------------------------------#
# Системные свойства FTP                                  #
#---------------------------------------------------------#

# Это системное свойство указывает, используется ли сокет многократно
# для нескольких операций передачи файлов (с помощью одного экземпляра
# FTP) в "активном" режиме.
# Если свойство равно true, значит сокет используется многократно. Если
# свойство равно faulse, значит для каждой передачи создается новый сокет.
# Это свойство игнорируется, если для объекта FTP была выполнена операция
# FTP.setReuseSocket(true/false)
com.ibm.as400.access.FTP.reuseSocket=true


#---------------------------------------------------------#
# Системное свойство соединения                           #
#---------------------------------------------------------#

# Данное системное свойство задает обработчик по умолчанию для входа в систему.
# Это свойство игнорируется, если для объекта AS400 была выполнена операция
# AS400.setSignonHandler() или вызвана функция
# AS400.setDefaultSignonHandler()
#
com.ibm.as400.access.AS400.signonHandler=mypackage.MyHandler


# Конец файла
```

Пример: Исходный файл класса системных свойств

```
//=========================================================
// IBM Toolbox for Java
//---------------------------------------------------------
// Исходный файл класса системных свойств
//
// Скомпилируйте этот файл и укажите файл класса
// в переменной CLASSPATH.
//=========================================================
package com.ibm.as400.access;

public class Properties
extends java.util.Properties
{
   public Properties ()
   {
      /*---------------------------------------------------------*/
      /* Системные свойства сервера Proxy                        */
      /*---------------------------------------------------------*/

      // Данное системное свойство задает имя хоста и номер порта
      // сервера Proxy в формате имя-хоста:номер-порта
      // Номер порта необязателен.
      put ("com.ibm.as400.access.AS400.proxyServer", "hqoffice");


      // Данное системное свойство указывает, с какой периодичностью
```

```
      // the proxy server will look for idle connections.       // Сервер Proxy запускает отдельную нить для поиска клие
      // не обменивающихся данными с сервером. Данное свойство
// позволяет задать частоту выполнения этой нити.
      put("com.ibm.as400.access.TunnelProxyServer.clientCleanupInterval", "7200");

      // Данное системное свойство задает время (в секундах)
      // простоя клиента перед удалением соединения. Сервер
// Proxy запускает отдельную нить для поиска клиентов,
      // не обменивающихся данными с сервером. Данное свойство
// позволяет задать время простоя соединения перед удалением.
      put("com.ibm.as400.access.TunnelProxyServer.clientLifetime", "2700");

      /*-------------------------------------------------------*/
      /* Свойства трассировки                                  */
      /*-------------------------------------------------------*/

      // Данное системное свойство задает применяемые категории трассировки.
      // В качестве значения можно указать список категорий через запятую.
      // Полный список категорий трассировки определен в классе
// Trace.
      put ("com.ibm.as400.access.Trace.category", "error,warning,information");

      // Данное системное свойство задает файл для записи вывода
      // трассировки. По умолчанию применяется файл System.out.
      put ("com.ibm.as400.access.Trace.file", "c:\temp\trace.out");


      /*-------------------------------------------------------*/
      /* Системные свойства вызова команд                      */
      /*-------------------------------------------------------*/

      // Это системное свойство указывает, поддерживают ли объекты
      // CommandCall нити. Если указано значение true - все объекты
// CommandCall считаются поддерживающими нити. Если указано false -
// все объекты считаются не поддерживающими нити. Это свойство
// игнорируется, если для объекта CommandCall был вызван метод
      // CommandCall.setThreadSafe(true/false) или
      // AS400.setMustUseSockets(true).
      put ("com.ibm.as400.access.CommandCall.threadSafe", "true");


      /*-------------------------------------------------------*/
      /* Системные свойства вызова программ                    */
      /*-------------------------------------------------------*/

      // Это системное свойство указывает, поддерживают ли объекты
      // ProgramCall нити. Если указано значение true - все объекты
// ProgramCall считаются поддерживающими нити. Если указано false -
// все объекты считаются не поддерживающими нити. Это свойство
// игнорируется, если для объекта ProgramCall был вызван метод
      // ProgramCall.setThreadSafe(true/false) или
      // AS400.setMustUseSockets(true).
      put ("com.ibm.as400.access.ProgramCall.threadSafe", "true");


      /*-------------------------------------------------------*/
      /* Системные свойства FTP                                */
      /*-------------------------------------------------------*/

      // Это системное свойство указывает, используется ли сокет многократно
      // для нескольких операций передачи файлов (с помощью одного экземпляра
      // в "активном" режиме. Если свойство равно true, значит сокет используется
      // многократно.
      // Если свойство равно false, значит для каждой передачи создается новый
      // сокет.
      // Это свойство игнорируется, если для объекта FTP была выполнена
      // операция FTP.setReuseSocket(true/false)
```

```
        put ("com.ibm.as400.access.FTP.reuseSocket", "true");


        /*-----------------------------------------------------------*/
        /* Системное свойство соединения                             */
        /*-----------------------------------------------------------*/

        // Данное системное свойство задает обработчик по умолчанию для входа
        //в систему.
        // Это свойство игнорируется, если для объекта AS400 была выполнена
        // операция AS400.setSignonHandler() или вызвана функция
        // AS400.setDefaultSignonHandler()
        //
        put ("com.ibm.as400.access.AS400.signonHandler", "mypackage.MyHandler");

    }
}
```

IBM Toolbox for Java classes

The IBM Toolbox for Java classes are categorized, like all Java classes, into packages. Каждый пакет предоставляет определенный набор функций.

Для удобства в настоящей документации всем пакетам присвоены краткие имена. Например, пакет com.ibm.as400.access называется просто пакетом доступа.

R In addition to the packages listed below, you can also read more about the micro package, which enables you to create
R Java programs that give your wireless devices direct access to System i5 data and services, in the “IBM Toolbox for Java 2 Micro Edition” на стр. 353 topic.

Классы доступа

The IBM Toolbox for Java access classes represent System i5 data and resources.

Примечание: IBM Toolbox for Java provides a second set of classes, called the resource classes, for working with System i5 objects and lists. The resource classes present a generic framework and consistent programming interface for working with various System i5 objects and lists.

Информация, связанная с данной

EventLog classes Javadoc
Provides a way to log exceptions and messages independent of the device used to display them.

Access package summary

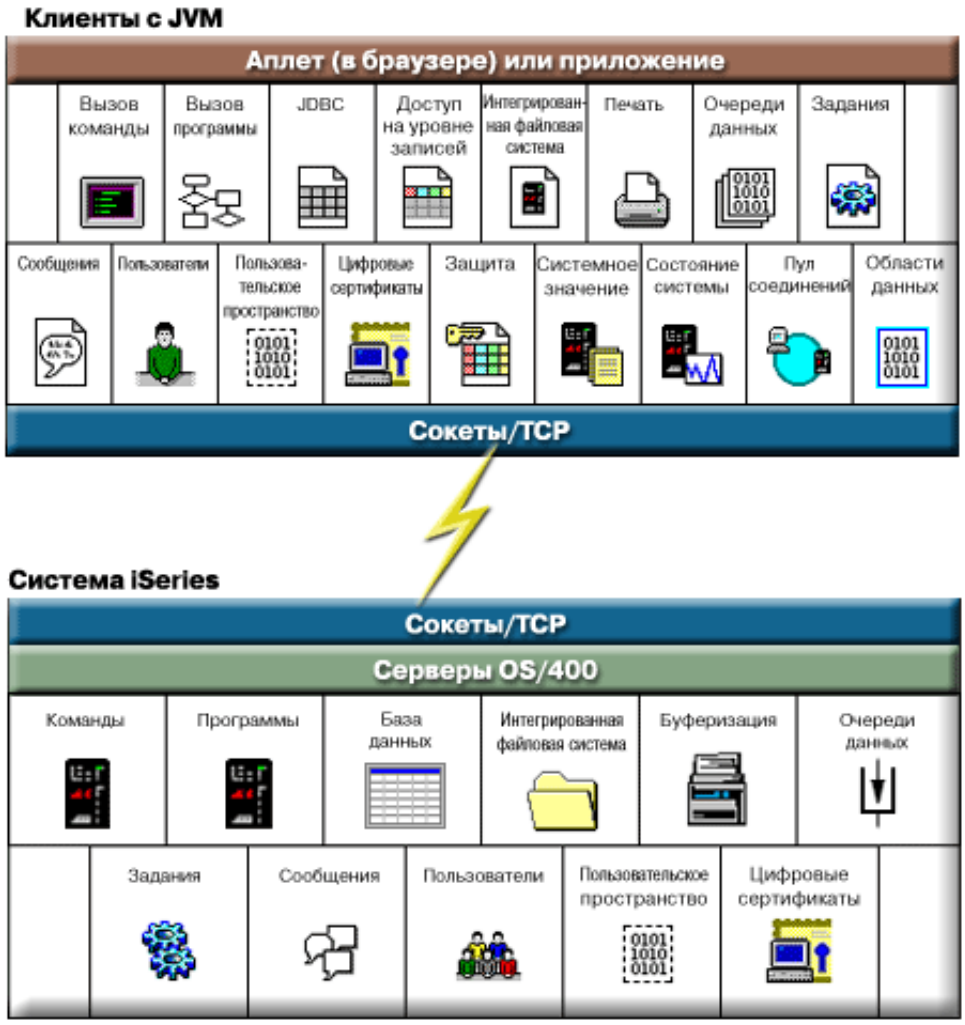
Resource package summary

Точки доступа сервера

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

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

Рис. 1: Точки доступа сервера



Класс AS400

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

R The Java program must provide an AS400 object when the Java program uses an instance of a class that accesses the
R System i5. For example, the CommandCall object requires an AS400 object before it can send commands to the
R system.

R The AS400 object handles connections, user IDs, and passwords differently when it is running in the i5/OS Java virtual
R machine. For more information, see “i5/OS Java virtual machine” на стр. 456.

Для объектов AS400 теперь поддерживается идентификация с помощью Kerberos: вместо применения ИД пользователя и пароля идентификация на сервере выполняется с помощью API Базовая служба защиты Java (JGSS).

Примечание: Для использования паспортов Kerberos необходимо установить J2SDK версии 1.4 и настроить интерфейс прикладных программ Базовая служба защиты Java (JGSS). For more information about JGSS, see the J2SDK, v1.4 Security Documentation .

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

Класс AS400 содержит следующие функции входа в систему:

- Authenticate the user profile
- Get a profile token credential and authenticate the associated user profile
- Set a profile token credential
- Управление идентификаторами пользователей по умолчанию
- Кэширование паролей
- Выдача приглашения для ввода идентификатора пользователя
- Change a password
- Get the version and release of the operating system

Информация об использовании объекта AS400 для обмена зашифрованными данными находится в разделе Класс SecureAS400.

Информация, связанная с данной

AS400ConnectionPool Javadoc

AS400 Javadoc

Управление идентификаторами пользователей по умолчанию:

Для сокращения числа операций входа в систему можно применять ИД пользователя по умолчанию. Программа на Java использует ИД по умолчанию в тех случаях, когда ИД пользователя не указан. ИД пользователя по умолчанию может быть задан программой Java или введен пользователем. Если ИД по умолчанию не задан, его можно установить с помощью окна диалога Вход в систему.

После установки идентификатора по умолчанию для сервера его изменение в окне Вход в систему невозможно. Во время создания объекта AS400 программа Java может передать конструктору ИД и пароль пользователя. При передаче идентификатора пользователя объекту AS400 ИД по умолчанию не изменяется. Для того чтобы задать или изменить в программе ИД пользователя по умолчанию, в ней необходимо явно указать этот ИД с помощью метода setUseDefaultUser(). Дополнительная информация приведена в разделе Сведения о приглашениях, идентификаторах пользователей по умолчанию и кэшировании паролей.

Объект AS400 включает методы для получения, установки и удаления идентификаторов пользователей по умолчанию. Программа на Java также может удалить идентификатор по умолчанию методом setUseDefaultUser(). If default user ID processing is disabled and the Java application does not supply a user ID, the AS400 object prompts for user ID every time a connection is made to the server.

All AS400 objects that represent the same System i5 within a Java virtual machine use the same default user ID.

В приведенном ниже примере два объекта AS400 используются для создания двух соединений с сервером. Если при входе в систему пользователь отметил опцию ИД пользователя по умолчанию, то при создании второго соединения приглашение не выдается.

```
                    // Создать два объекта AS400
                    // same system.
AS400 sys1 = new AS400("mySystem.myCompany.com");
AS400 sys2 = new AS400("mySystem.myCompany.com");

                    // Установить соединение со службой
                    // вызова команд. Будет показано приглашение для ввода
                    // идентификатора пользователя и пароля.
sys1.connectService(AS400.COMMAND);
```

```
                    // Установить второе соединение
                    // со службой вызова команд.                    // Приглашение показано не будет.
sys2.connectService(AS400.COMMAND);
```

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

Применение кэша паролей:

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

Методы объекта AS400 позволяют выполнить следующие действия:

- Очистить кэш паролей и отключить кэш паролей
- Сократить число повторных вводов информации для входа в систему

R The password cache applies to all AS400 objects that represent a System i5 within a Java virtual machine. Java не
R обеспечивает общего использования данных несколькими виртуальными машинами, поэтому пароли,
R находящиеся в кэше одной виртуальной машины Java, недоступны остальным виртуальным машинам. Кэш
R очищается во время сбора мусора при удалении последнего объекта AS400. Пользователь может включить
R кэширование пароля с помощью переключателя в окне диалога Вход в систему. Программы Java могут
R передавать ИД и пароль пользователя во время создания объекта AS400. Пароли, передаваемые
R конструктору, не кэшируются.

Объект AS400 включает методы для очистки кэша паролей и отключения кэша паролей. Дополнительная информация приведена в разделе Сведения о приглашениях, идентификаторах пользователей по умолчанию и кэшировании паролей.

Приглашение для ввода имени пользователя и пароля:

When you are using the AS400 class, prompting for user ID and password may occur when connecting to the server. Может быть отключен программой на Java

Программы на Java +могут отключить выдачу приглашения для ввода идентификатора пользователя и пароля, а также других сообщений объекта AS400. Это может потребоваться, например, в том случае, если приложение работает на шлюзе, выступая от имени нескольких клиентов. Очевидно, пользователи не смогут в интерактивном режиме реагировать на приглашения и сообщения, появляющиеся на шлюзе. Вывод приглашений может быть отключен методом setGuiAvailable() объекта AS400.

Дополнительная информация приведена в разделе Сведения о приглашениях, идентификаторах пользователей по умолчанию и кэшировании паролей.

Сведения о приглашениях, идентификаторах пользователей по умолчанию и кэшировании паролей:

Программы на Java могут управлять выводом приглашений и кэшированием паролей. Информация, вводимая в окне диалога Вход в систему, может применяться для установки идентификатора пользователя по умолчанию и кэширования паролей. В следующей таблице собрана информация о том, когда выдаются приглашения, какие сведения необходимо указывать и какие параметры устанавливаются.

Предполагается, что в программе Java разрешено применение идентификатора пользователя по умолчанию и кэширование паролей, а в окне диалога Вход в систему были отмечены опции **ИД пользователя по умолчанию** и **Сохранить пароль**.

Данные в этой таблице служат для создания соединений с клиентами, а не для запуска Java на сервере.

Система, указанная в конструкторе	ИД пользователя, указанный в конструкторе	Пароль, указанный в конструкторе	Установлен пользователь по умолчанию	Пароль в кэше для ИД пользователя	Результат применения отмеченных параметров
					Выдается приглашение, в котором пользователь должен указать имя системы, свой идентификатор и пароль. Устанавливается идентификатор пользователя по умолчанию; пароль помещается в кэш.
Да					Выдается приглашение, в котором пользователь должен указать свое имя и пароль. Имя системы показано, но не может быть изменено. Устанавливается идентификатор пользователя по умолчанию; пароль помещается в кэш.
Да	**Да**				Выдается приглашение, в котором пользователь должен указать пароль. ИД пользователя показан и может быть изменен. Имя системы показано, но не может быть изменено. Идентификатор пользователя по умолчанию не изменяется. Пароль помещается в кэш.

Система, указанная в конструкторе	ИД пользователя, указанный в конструкторе	Пароль, указанный в конструкторе	Установлен пользователь по умолчанию	Пароль в кэше для ИД пользователя	Результат применения отмеченных параметров
Да	**Да**	**Да**			Приглашение не выдается. Идентификатор пользователя по умолчанию не изменяется. Пароль не сохраняется в кэше.
			Да		Выдается приглашение, в котором пользователь должен указать имя системы и пароль. ИД пользователя показан и может быть изменен. Изменение идентификатора не повлечет изменения идентификатора по умолчанию. Пароль помещается в кэш.
Да			**Да**		Выдается приглашение, в котором пользователь должен ввести идентификатор по умолчанию. ИД пользователя показан и может быть изменен. Имя системы показано, но не может быть изменено. Пароль помещается в кэш.
Да			**Да**	**Да**	Приглашение не выдается. При подключении применяются ИД пользователя по умолчанию и пароль из кэша.

Система, указанная в конструкторе	ИД пользователя, указанный в конструкторе	Пароль, указанный в конструкторе	Установлен пользователь по умолчанию	Пароль в кэше для ИД пользователя	Результат применения отмеченных параметров
Да	**Да**			**Да**	Приглашение не выдается. При подключении применяются указанный ИД пользователя и пароль из кэша.
Да	**Да**		**Да**	**Да**	Приглашение не выдается. При подключении применяются указанный ИД пользователя и пароль из кэша.
Да	**Да**	**Да**	**Да**		Приглашение не выдается. При подключении применяется указанный ИД пользователя.

Класс SecureAS400

The SecureAS400 class enables you to use an AS400 object when sending or receiving encrypted data. Все пользовательские данные (за исключением пароля пользователя) передаются от объекта AS400 на сервер в незашифрованном виде. Следовательно, все объекты IBM Toolbox for Java, связанные с объектом AS400, обмениваются данными с сервером по обычному соединению.

Для передачи по сети секретных данных IBM Toolbox for Java их можно зашифровать с помощью Secure Sockets Layer (SSL). Use the SecureAS400 object to designate which data you want to encrypt. Объекты IBM Toolbox for Java, связанные с объектом SecureAS400, обмениваются данными с сервером по защищенному соединению.

Дополнительная информация приведена в разделе Secure Sockets Layer и Java Secure Socket Extension.

The SecureAS400 class is a subclass of the AS400 class.

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

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

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

```
      // Создание защищенного объекта AS400. Это единственный оператор,
// который требуется добавить в случае SSL.
     SecureAS400 sys = new SecureAS400("mySystem.myCompany.com");

     // Создание объекта вызова команды
     CommandCall cmd = new CommandCall(sys, "myCommand");

     // Запуск команд.  При выполнении команды создается
     // защищенное соединение. После этого клиент и сервер
// обмениваются зашифрованной информацией.
     cmd.run();
```

Информация, связанная с данной

SecureAS400 Javadoc

AS400 Javadoc

Класс AS400JPing

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

Такой запрос можно отправить из командной строки с помощью класса JPing.

Класс AS400JPing содержит следующие методы:

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

Пример: Применение класса AS400JPing

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

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

Информация, связанная с данной

Класс AS400JPing

Класс BidiTransform

The IBM Toolbox for Java BidiTransform class provides layout transformations that enable you to convert bidirectional text in i5/OS format (after first converting it to Unicode) to bidirectional text in Java format, or from Java format to i5/OS format.

Класс AS400BidiTransform

Класс AS400BidiTransform позволяет:

- Get and set the system CCSID
- Get and set the string type of i5/OS data
- Get and set the string type of Java data
- Convert data from a Java layout to i5/OS
- Convert data from an i5/OS layout to Java

Пример: Применение класса AS400BidiTransform

Следующий пример иллюстрирует преобразование двунаправленного текста с помощью класса AS400BidiTransform:

```
// Java data to i5/OS layout:
AS400BidiTransform abt;
abt = new AS400BidiTransform(424);
String dst = abt.toAS400Layout("некоторая двунаправленная строка");
```

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

BidiConversionProperties

Свойства класса BidiConversionProperties позволяют управлять преобразованием данных в разных кодировках.

Информация, связанная с данной

BidiConversionProperties Javadoc

CallStackEntry

Представляет запись в стеке вызовов отдельной нити или задания сервера.

Объекты этого типа генерируются при вызове `Job.getCallStack()`.

Информация, связанная с данной

CallStackEntry Javadoc

Классы ClusteredHashTable

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

Перед началом работы с классами ClusteredHashTable убедитесь, что вы можете сохранять данные в непостоянной памяти. Копируемые данные не шифруются.

R **Примечание:** The following information assumes that you understand the concepts and terminology common to i5/OS
R cluster technology. See i5/OS cluster technology for details.

Using the ClusteredHashTable class requires that you have defined and activated a cluster on the systems. Кроме того, необходимо запустить сервер кластерных хэш-таблиц. Дополнительная информация приведена в разделе Настроить кластеры и API кластерных хэш-таблиц.

Обязательные параметры - это имя сервера кластерных хэш-таблиц, а также объект AS400, представляющий систему, в которой находится этот сервер кластерных хэш-таблиц.

Для хранения данных на сервере кластерных хэш-таблиц необходимы описатель и ключ соединения:

- Когда вы открываете соединение, сервер кластерных хэш-таблиц присваивает описатель соединения, который вы должны указать в последующих запросах к этому серверу. Этот описатель соединения пригоден только для данного объекта AS400; если вы хотите использовать другой объект AS400, то вы должны открыть новое соединение.
- Вы должны задать ключ для доступа и изменения данных в кластерной хэш-таблице. Ключ должен быть уникальным.

Методы класса ClusteredHashTable позволяют выполнить следующие действия:

- Открыть соединение с заданием сервера кластерных хэш-таблиц
- Создать уникальный ключ для хранения данных в кластерной хэш-таблице
- Закрыть активное соединение с заданием сервера кластерных хэш-таблиц

Некоторые методы класса ClusteredHashTable применяют класс ClusteredHashTableEntry для выполнения следующих действий:

- Получить запись из кластерной хэш-таблицы
- Сохранить запись в кластерной хэш-таблице
- Получить список записей из кластерной хэш-таблицы для всех пользовательских профайлов

Пример: Использование ClusteredHashTable

В следующем примере рассмотрен сервер кластерных хэш-таблиц CHTSVR01. Предполагается, что кластер и сервер кластерных хэш-таблиц уже активны. Сервер открывает соединение, создает ключ, с помощью этого ключа помещает запись в кластерную хэш-таблицу, получает запись из этой таблицы и закрывает соединение.

```
ClusteredHashTableEntry myEntry = null;

String myData = new String("Это мои данные.");
System.out.println("Данные для сохранения: " + myData);

AS400 system = new AS400();

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

// Открытие соединения.
cht.open();

// Получение ключа к хэш-таблице
byte[] key = null;
key = cht.generateKey();

// Подготовка данных для сохранения в кластерной хэш-таблице.
// ENTRY_AUTHORITY_ANY_USER означает, что записи
// кластерной хэш-таблице доступны любому пользователю.
// DUPLICATE_KEY_FAIL означает, что если заданный ключ уже существует,
// то запрос ClusteredHashTable.put() выполнен не будет.
int timeToLive = 500;
myEntry = new ClusteredHashTableEntry(key,myData.getBytes(),timeToLive,
     ClusteredHashTableEntry.ENTRY_AUTHORITY_ANY_USER,
     ClusteredHashTableEntry.DUPLICATE_KEY_FAIL);

// Сохранение (или размещение) записи в хэш-таблице.
cht.put(myEntry);

// Получение записи из хэш-таблицы.
ClusteredHashTableEntry output = cht.get(key);

// Закрытие соединения.
cht.close();
```

При использовании класса ClusteredHashTable объект AS400 автоматически подключается к серверу. Дополнительная информация приведена в разделе Управление соединениями.

CommandCall - Access package

The CommandCall class allows a Java program to call a non-interactive System i5 command.

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

Для работы CommandCall необходимо следующее:

- Текст исполняемой команды
- The AS400 object that represents the system that will run the command

R The command string can be set on the constructor, through the CommandCall setCommand() method, or on the run()
R method. After the command is run, the Java program can use the getMessageList() method to retrieve any System i5
R messages resulting from the command.

Using the CommandCall class causes the AS400 object to connect to the system. Информация об управлении соединениями приведена в разделе управление соединениями.

R When the Java program and the System i5 command are on the same server, the default IBM Toolbox for Java behavior
R is to look up the thread safety for the command on the system. В случае положительного результата команда
R запускается в отдельной нити, а не в процессе. You can suppress the run-time lookup by explicitly specifying
R thread-safety for the command by using the setThreadSafe() method.

Примеры

Ниже приведены примеры запуска разных типов команд с помощью класса CommandCall.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Пример: Запуск команды

The following example shows how to use the CommandCall class to run a command on the system:

```
                    // Создание объекта AS400
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта вызова команды.                        // В этой программе команда будет
                    // определена позднее. Она могла быть
                    // определена в конструкторе.
CommandCall cmd = new CommandCall(sys);

                    // Запуск команды CRTLIB
cmd.run("CRTLIB MYLIB");

                    // Получение списка сообщений
                    // о результатах выполнения
                    // команды.
AS400Message[] messageList = cmd.getMessageList();

                    // ... обработка списка сообщений.

                    // Отсоединение по окончании
                    // отправки команд на сервер
sys.disconnectService(AS400.COMMAND);
```

Пример: Запуск пользовательской команды

Раздел “Пример: применение объекта CommandCall” на стр. 475 содержит пример запуска пользовательской команды.

Информация, связанная с данной

CommandCall Javadoc

AS400Message Javadoc

AS400 Javadoc

Пул соединений

Use connection pools to share connections and manage sets (pools) of connections to a System i5. Например, приложение может получить соединение из пула, воспользоваться им, а затем возвратить его обратно в пул для дальнейшего применения.

The AS400ConnectionPool class manages a pool of AS400 objects. The AS400JDBCConnectionPool class represents a pool of AS400JDBCConnections that are available for use by a Java program as part of IBM Toolbox for Java support for the JDBC 2.0 Optional Package API. Интерфейс ConnectionPool JDBC также поддерживается в API JDBC 3.0, встроенном в платформу Java 2, Standard Edition, версии 1.4.

Пул соединений любого типа отслеживает число создаваемых соединений. Using methods inherited from ConnectionPool, you can set several connection pool properties, including:

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

С точки зрения быстродействия подключение к серверу - дорогостоящая операция. Пулы соединений позволяют повысить быстродействие за счет того, что повторное подключение отдельного соединения не требует дополнительного времени. For example, create connections when you create the connection pool by filling the pool with active (preconnected) connections using the AS400ConnectionPool class. Вместо того чтобы создавать новые соединения, вы можете получать, использовать, возвращать и вновь использовать уже существующие соединения из пула.

Для того чтобы получить соединение из пула AS400ConnectionPool, необходимо указать имя системы, ИД пользователя, пароль и службу (необязательно). To specify the service to which you want to connect, use constants from the AS400 class (FILE, PRINT, COMMAND, and so on).

По окончании работы с соединениями, полученными из пула, приложения должны возвращать соединения в пул. Учтите, что ответственность за возврат соединений в пул для повторного использования лежит на конкретном приложении. Если соединения не возвращаются в пул, то размер пула растет, а соединения не используются повторно.

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

Пример: Применение AS400ConnectionPool

“Пример: Применение AS400ConnectionPool” на стр. 476 содержит информацию о повторном использовании объектов AS400.

Информация, связанная с данной

AS400ConnectionPool Javadoc

AS400 Javadoc

Область данных

The IBM Toolbox for Java DataArea class is an abstract base class that represents a System i5 data area object.

На основе этого класса определены четыре подкласса, соответствующие символьным данным, десятичным данным, логическим данным и локальным областям символьных данных.

Класс DataArea позволяет выполнять следующие операции:

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

При использовании класса DataArea объект AS400 автоматически подключается к серверу. Информация об управлении соединениями приведена в разделе управление соединениями.

CharacterDataArea

The CharacterDataArea class represents a data area on the server that contains character data. Области символьных данных не имеют средств регистрации CCSID хранимых данных, поэтому объект области данных считает, что данные используют пользовательский CCSID. При записи информации в область данных она преобразуется из строки формата Unicode в пользовательский CCSID. При чтении объект предполагает, что

данные закодированы на основе пользовательского CCSID, и перекодирует их в Unicode перед возвратом вызывающей программе. При чтении строк из области данных объем требуемой информации задается количеством символов, а не числом байт.

Класс CharacterDataArea позволяет выполнять следующие операции:

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

DecimalDataArea

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

Класс DecimalDataArea позволяет выполнять следующие операции:

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

Пример: Применение класса DecimalDataAreaНиже приведен пример создания и записи в область десятичных данных:

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код примеров.

```
   // Establish a connection to the server "MyServer".
AS400 system = new AS400("MyServer");
   // Создание объекта DecimalDataArea.
QSYSObjectPathName path = new QSYSObjectPathName("MYLIB", "MYDATA", "DTAARA");
DecimalDataArea dataArea = new DecimalDataArea(system, path.getPath());
   // Создание на сервере области десятичных данных со свойствами по умолчанию.
dataArea.create();
   // Очистка области данных.
dataArea.clear();
   // Запись значения в созданную область данных.
dataArea.write(new BigDecimal("1.2"));
   // Получение значения из области данных.
BigDecimal data = dataArea.read();
   // Удаление области данных с сервера.
dataArea.delete();
```

LocalDataArea

The LocalDataArea class represents a local data area on the server. Локальная область данных определяется на сервере как область символьных данных, однако при работе с ней следует учитывать ряд ограничений.

Локальная область данных относится к заданию сервера; доступ к ней из других заданий запрещен. По этой причине, локальную область данных нельзя ни создать, ни удалить. При завершении задания связанная с ним локальная область данных автоматически удаляется, а объект LocalDataArea, ссылающийся на такую область, становится недействительным. Размер локальных областей данных на сервере фиксирован и равен 1024 символам.

Класс LocalDataArea позволяет выполнять следующие операции:

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

LogicalDataArea

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

Класс LogicalDataArea позволяет выполнять следующие операции:

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

DataAreaEvent

The DataAreaEvent class represents a data area event.

Класс DataAreaEvent может применяться со всеми классами семейства DataArea. Класс DataAreaEvent позволяет выполнять следующие операции:

- Get the identifier for the event

DataAreaListener

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

Класс DataAreaListener может применяться со всеми классами семейства DataArea. Класс DataAreaListener может быть активизирован по любой из следующих операций:

- Clear
- Create
- Delete
- Read
- Запись

DataArea Javadoc

CharacterDataArea Javadoc

DecimalDataArea Javadoc

LocalDataArea Javadoc

LogicalDataArea Javadoc

DataAreaEvent Javadoc

DataAreaListener Javadoc

Преобразование и описание данных

The data conversion classes provide the capability to convert numeric and character data between i5/OS and Java formats. Conversion may be needed when accessing i5/OS data from a Java program. Предусмотрены классы для преобразования между различными числовыми форматами и между кодовыми страницами EBCDIC и Unicode.

R Классы **описания данных** построены на базе классов преобразования данных и предназначены для
R преобразования всех полей записи путем вызова одного метода. The RecordFormat class allows the program to
R describe data that makes up a DataQueueEntry, ProgramCall parameter, a record in a database file accessed through
R record-level access classes, or any buffer of system data. Класс Record позволяет преобразовывать содержимое
R всей записи и обращаться к отдельным полям записи по имени или индексу поля.

The **converter** classes provide fast and efficient conversion between Java and your system. Класс BinaryConverter преобразует массивы байтов Java в простые типы Java и обратно. CharConverter converts between Java String objects and i5/OS code pages. For more information, see the Converters topic.

Типы данных

The AS400DataType is an interface that defines the methods required for data conversion. Эти методы реализуются в программе на Java при преобразовании конкретных типов данных. Существуют классы преобразования для следующих типов данных:

- Числовые типы
- Текстовые (символьные) типы
- Составные типы

Пример: Применение классов AS400DataType

Ниже приведен пример того, как классы AS400DataType применяются с объектом ProgramCall для передачи параметров программе и для интерпретации параметров, возвращенных программой.

Пример: Использование классов AS400DataType с ProgramCall

Преобразование с указанием формата записи

R IBM Toolbox for Java позволяет создавать классы для преобразования целых записей, а не отдельных полей
R данных. Например, пусть программа на Java получает данные из очереди данных. The data queue object returns
R a byte array of i5/OS data to the Java program. This array can potentially contain many types of i5/OS data.
R Приложение, вместо того чтобы преобразовывать эти данные по одному полю, может создать формат
R записи, описывающий поля массива. Этот формат будет обеспечивать преобразование всех данных записи
R одним вызовом метода.

Преобразование с помощью формата записи полезно при работе с данными, полученными в результате вызова программы, из очереди данных или классов доступа на уровне записей. Ввод и вывод при этом представляет собой двоичный массив, который может включать множество полей различных типов. Record format converters can make it easier to convert this data between i5/OS format and Java format.

При преобразовании записей применяются классы трех типов:

- Классы FieldDescription связывают с полем или параметром имя и тип данных.
- Класс RecordFormat описывает группу полей.
- Класс Record объединяет описание (класс RecordFormat) и фактические данные записи.
- Класс LineDataRecordWriter добавляет запись в OutputStream в формате строковых данных

Пример: Применение классов преобразования форматов записей

Ниже приведен пример использования классов преобразования форматов записей при работе с очередями данных:

Запись данных в очередь с помощью классов Record и RecordFormat

Применение классов FieldDescription, RecordFormat и Record
AS400DataType Javadoc

Классы преобразования для числовых данных:

Conversion classes for numeric data convert numeric data from the format used on the System i5 (called **system format** in the following table) to the Java format.

Поддерживаемые типы перечислены в следующей таблице:

Числовой тип	**Описание**
AS400Bin2	Converts between a signed two-byte number in the system format to a Java Short object.
AS400Bin4	Converts between a signed four-byte number in the system format and a Java Integer object.
AS400ByteArray	Преобразует один массив байт в другой. Этот преобразование применяется для правильного дополнения целевого буфера нулями.
AS400Float4	Converts between a signed four-byte floating point number in the system format and a Java Float object.
AS400Float8	Converts between a signed eight-byte floating point number in the system format and a Java Double object.
AS400PackedDecimal	Converts between a packed-decimal number in the system format and a Java BigDecimal object.
AS400UnsignedBin2	Converts between an unsigned two-byte number in the system format and a Java Integer object.
AS400UnsignedBin4	Converts between an unsigned four-byte number in the system format and a Java Long object.
AS400ZonedDecimal	Converts between a zoned-decimal number in the system format and a Java BigDecimal object.

Примеры

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

Example: Converting from the system format to a Java int

```
// Create a buffer to hold the system data type. Предполагается, что в буфере
// filled with numeric data in the system format by data queues,
// вызовы программ и т.д.
```

```
byte[] data = new byte[100];

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

// Convert from system type to Java object. Число расположено в начале
// буфера.
Integer intObject = (Integer) bin4Converter.toObject(data,0);

// Извлечение простого типа Java из объекта Java.
int i = intObject.intValue();
```

Example: Converting from a Java int to the system format

```
// Создание объекта Java, который содержит преобразуемое значение.
Integer intObject = new Integer(22);

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

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

// Определение части буфера, которая была занята значением
// system value.
int length = bin4Converter.getByteLength();
```

Преобразование текста:

Character data is converted through the IBM Toolbox for Java AS400Text class. Этот класс преобразует текст между кодовой страницей EBCDIC с заданным CCSID и кодировкой Unicode.

When the AS400Text object is constructed, the Java program specifies the length of the string to be converted and the server CCSID or encoding. Предполагается, что в программе на Java используется CCSID 13488 Unicode. The toBytes() method converts from Java form to byte array ini5/OS format. The toObject() method converts from a byte array in i5/OS format to Java format.

The AS400BidiTransform class provides layout transformations that allow the conversion of bidirectional text in i5/OS format (after its conversion to Unicode) to bidirectional text in Java format, or from Java format to i5/OS format. По умолчанию преобразование выполняется с учетом CCSID задания. To alter the direction and shaping of the text, specify a BidiStringType. Note that where IBM Toolbox for Java objects perform the conversion internally, as in the DataArea class, the objects have a method to override the string type. В классе DataArea предусмотрен метод addVetoableChangeListener(), позволяющий отслеживать запрещенные изменения некоторых свойств, в том числе типа строки.

Пример: Преобразование текстовых данных

В приведенном ниже примере предполагается, что объект DataQueueEntry возвращает текст в кодировке EBCDIC. В примере данные EBCDIC преобразуются в Unicode, позволяющий применять их в программе на Java:

```
R      // Предполагается, что записи очереди данных уже извлечены из системы
R      // retrieve the text from the system and the data has been
R      // в следующий буфер.
R      int textLength = 100;
R      byte[] data = new byte[textLength];
R
R      // Create a converter for the system data type. Note a default
R      // converter is being built.  This converter assumes the System i5
R      // страница EBCDIC на сервере iSeries совпадает с локалью клиента. В
R // противном случае, в программе можно явно указать CCSID EBCDIC.
R      // Рекомендуется по возможности всегда указывать CCSID.
R // (см. Примечания).
```

```
R       AS400Text textConverter = new AS400Text(textLength)
R
R       // Примечание: Объект-преобразователь можно создать для конкретного
R       // CCSID. Если программа работает как клиент proxy Toolbox for Java,
R  // необходимо использовать объект AS400.
R       int ccsid = 37;
R       AS400 system = ...; // Объект AS400
R       AS400Text textConverter = new AS400Text(textLength, ccsid, system);
R
R       // Примечание: Можно создать объект-преобразователь с помощью одного
R       // This converter assumes the System i5 code page matches
R       // совпадает с CCSID, возвращенным объектом AS400.
R       AS400Text textConverter = new AS400Text(textLength, system);
R
R       // Преобразование данных из формата EBCDIC в Unicode. Если длина
R  // объекта AS400Text превышает число преобразуемых
R       // символов, полученный объект String будет
R       // дополнен пробелами до указанной длины.
R       String javaText = (String) textConverter.toObject(data);
```

Информация, связанная с данной

AS400Text Javadoc

AS400BidiTransform Javadoc

BidiStringType Javadoc

Классы преобразования для составных типов:

This topic describes IBM Toolbox for Java conversion classes for composite types.

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

Пример: Преобразование составных типов данных

В следующем примере показано преобразование структуры Java в двоичный массив системы AS/400 и обратно. Предполагается, что для приема и отправки данных применяется один и тот же формат.

```
R       // Создание структуры типов данных, соответствующей структуре, которая
R       // содержит: - четырехбайтовый номер
R       //    - четыре байта выравнивания
R       //    - восьмибайтовое число
R       //    - 40 символов
R       AS400DataType[] myStruct =
R       {
R          new AS400Bin4(),
R          new AS400ByteArray(4),
R          new AS400Float8(),
R          new AS400Text(40)
R       };
R
R       // Создание объекта преобразования с помощью структуры.
R       AS400Structure myConverter = new AS400Structure(myStruct);
R
R       // Создание объекта Java, содержащего данные для отправки на сервер.
R       Object[] myData =
R       {
R          new Integer(88),        // четырехбайтовое число
R          new byte[0],            // выравнивание (здесь - нулевого размера)
R          new Double(23.45),      // восьмибайтовое действительное число
R          "This is my structure"  // строка символов
R       };
R
R       // Преобразование объекта Java в двоичный массив.
R       byte[] myAS400Data = myConverter.toBytes(myData);
R
```

```
R
R       // ... отправка массива байтов на сервер. Получение данных с
R       // сервера. Полученные данные также будут массивом байтов.
R
R
R       // Convert the returned data from System i to Java format.
R       Object[] myRoundTripData = (Object[])myConverter.toObject(myAS400Data,0);
R
R       // Получение третьего объекта структуры. Объект имеет тип double.
R       Double doubleObject = (Double) myRoundTripData[2];
R
R       // Извлечение простого типа Java из объекта Java.
R       double d = doubleObject.doubleValue();
```

AS400Array Javadoc

AS400Structure Javadoc

Классы FieldDescription:

Классы описания полей позволяют программам на Java задавать описания полей, содержащие тип данных и имя поля. If the program is working with data from record-level access, it can also specify any i5/OS data definition specification (DDS) keywords that further describe the field.

Описание

Определены следующие классы описания полей:

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

Пример: Создание описаний полей

В приведенном ниже примере предполагается, что записи в очереди данных имеют один формат. Каждая запись состоит из номера сообщения (типа AS400Bin4), значения времени (8 символов) и текста сообщения (50 символов), для которых можно создать описания полей:

```
// Создание описания поля для числовых данных. В нем используется
// тип данных AS400Bin4. Кроме того, описание задает имя поля, с помощью которого поле
// можно найти в классе записей.
BinaryFieldDescription bfd = new BinaryFieldDescription(new AS400Bin4(), "msgNumber");

// Создание описания поля для символьных данных. В нем используется
// тип данных AS400Text. Кроме того, описание задает имя поля, с помощью которого поле
// можно найти в классе записей.
CharacterFieldDescription cfd1 = new CharacterFieldDescription(new AS400Text(8), "msgTime");

// Создание описания поля для символьных данных. В нем используется
```

```
// тип данных AS400Text. Кроме того, описание задает имя поля, с помощью которого поле
// можно найти в классе записей.
CharacterFieldDescription cfd2 = new CharacterFieldDescription(new AS400Text(50), "msgText");
```

Полученные описания полей можно объединить в экземпляре класса RecordFormat. Пример такого объединения приведен на следующей странице:

“Класс RecordFormat”

Класс RecordFormat:

The IBM Toolbox for Java RecordFormat class allows the Java program to describe a group of fields or parameters. Данные, описанные объектом RecordFormat, можно поместить в объект записи. При работе на уровне записей класс RecordFormat также позволяет программе указать описания ключевых полей.

Объект RecordFormat представляет собой набор описаний полей. Доступ к этим описаниям возможен по индексу или по имени. Класс RecordFormat содержит следующие методы:

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

Пример: Добавление описаний полей в формат записи

В приведенном ниже примере описания полей, созданные в примере описаний полей, добавляются в формат записи:

```
// Создание объекта формата записи и добавление в него описаний полей.
RecordFormat rf = new RecordFormat();
rf.addFieldDescription(bfd);
rf.addFieldDescription(cfd1);
rf.addFieldDescription(cfd2);
```

Пример создания записи с помощью объекта формата записи приведен на следующей странице:

“Класс Record”
RecordFormat Javadoc

Класс Record:

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

При этом выполняется преобразование данных из двоичных массивов сервера в объекты Java и обратно. Класс записи включает следующие методы:

- Получение содержимого поля, выбранного по индексу или имени, в виде объекта Java.
- Получение числа полей записи.
- Задание содержимого поля, выбранного по индексу или имени, с помощью объекта Java.
- Получение содержимого записи в виде двоичного массива или потока вывода данных сервера.
- Создание содержимого записи из массива двоичных данных или потока ввода.

- Преобразование содержимого записи в строку (объект типа String).

Пример: Считывание записи

Ниже приведен пример использования формата записи, созданного в примере формат записи:

```
// Предполагается, что очередь данных уже настроена.      // Чтение из очереди данных:
DataQueueEntry dqe = dq.read();

// Данные из очереди данных находятся в записи этой очереди. Эти данные
// извлекаются из записи и помещаются в объект записи.
// Объект записи по умолчанию создается с помощью объекта формата записи и
// инициализируется с данными из записи очереди данных.
Record dqRecord = rf.getNewRecord(dqe.getData());

// После занесения данных в объект записи их необходимо
// извлечь из поля и преобразовать перед его удалением. Результатом
// будет объект Java с данными, который можно передавать в программу.
Integer msgNumber = (Integer) dqRecord.getField("msgNumber");
String  msgTime   = (String)  dqRecord.getField("msgTime");
String  msgText   = (String)  dqRecord.getField("msgText");
```

Информация, связанная с данной

Record Javadoc

Получение содержимого поля:

В программе Java может потребоваться получить как одно поле объекта Record, так и все его поля сразу.

Use the getField() method of the Record class to retrieve a single field by name or by index. Для получения содержимого всех полей в виде массива объектов предназначен метод getFields().

В программе Java полученный объект Java (или элемент массива объектов) необходимо преобразовать к соответствующему типу. Объекты Java, соответствующие полям различных типов, перечислены в следующей таблице.

DDS поля	Значение FieldDescription поля	Объект Java
Двоичные данные (B), длина <= 4	BinaryFieldDescription	Short
BINARY (B), length >= 5	BinaryFieldDescription	Integer
Текст (A)	CharacterFieldDescription	String
Либо-DBCS (E)	DBCSEitherFieldDescription	String
Графический-DBCS (G)	DBCSGraphicFieldDescription	String
Только-DBCS (J)	DBCSOnlyFieldDescription	String
Открытый-DBCS (O)	DBCSOpenFieldDescription	String
Дата (L)	DateFieldDescription	String
Число с плавающей точкой (F) одинарной точности	FloatFieldDescription	Float
Число с плавающей точкой (F) двойной точности	FloatFieldDescription	Double
Шестнадцатеричные данные (H)	HexFieldDescription	byte[]
Упакованные десятичные данные (P)	PackedDecimalFieldDescription	BigDecimal
Время (T)	TimeDecimalFieldDescription	String
Системное время (Z)	TimestampDecimalFieldDescription	String
Зонные десятичные данные (P)	ZonedDecimalFieldDescription	BigDecimal

Информация, связанная с данной

Record Javadoc

Задание содержимого поля:

В программе на Java содержимое объекта Record можно задать с помощью метода setField().

Значение поля в программе Java задается с помощью соответствующего объекта Java. Объекты Java, соответствующие полям различных типов, перечислены в следующей таблице.

DDS поля	Значение FieldDescription поля	Объект Java
Двоичные данные (B), длина <= 4	BinaryFieldDescription	Short
BINARY (B), length >= 5	BinaryFieldDescription	Integer
Текст (A)	CharacterFieldDescription	String
Либо-DBCS (E)	DBCSEitherFieldDescription	String
Графический-DBCS (G)	DBCSGraphicFieldDescription	String
Только-DBCS (J)	DBCSOnlyFieldDescription	String
Открытый-DBCS (O)	DBCSOpenFieldDescription	String
Дата (L)	DateFieldDescription	String
Число с плавающей точкой (F) одинарной точности	FloatFieldDescription	Float
Число с плавающей точкой (F) двойной точности	FloatFieldDescription	Double
Шестнадцатеричные данные (H)	HexFieldDescription	byte[]
Упакованные десятичные данные (P)	PackedDecimalFieldDescription	BigDecimal
Время (T)	TimeDecimalFieldDescription	String
Системное время (Z)	TimestampDecimalFieldDescription	String
Зонные десятичные данные (P)	ZonedDecimalFieldDescription	BigDecimal

Информация, связанная с данной

Record Javadoc

Класс LineDataRecordWriter:

Класс LineDataRecordWriter заносит данные записи в строковом формате в OutputStream. Этот класс преобразует данные в поток байт с учетом указанного CCSID. Формат данных определяется форматом записи.

LineDataRecordWriter

Для применения LineDataRecordWriter необходимо задать следующие атрибуты формата записи:

- ИД формата записи
- Тип формата записи

In conjunction with the Record or the RecordFormat classes, the LineDataRecordWriter takes a record as input to the writeRecord() method. (При создании экземпляра записи указывается RecordFormat.)

Класс LineDataRecordWriter содержит методы, позволяющие:

- Get the CCSID
- Get the name of the encoding

- Write the record data, in line data format, to an OutputStream

Пример: Применение класса LineDataRecordWriter

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

В следующем примере показан один из способов занесения записи в очередь сообщений с помощью класса LineDataRecordWriter:

```
// Пример применения класса LineDataRecordWriter.
   try
   {
     // создание ccsid
               ccsid_ = system_.getCcsid();

     // создание очереди вывода и настройка значения *LINE в качестве формата буферного файла
               OutputQueue outQ = new OutputQueue(system_, "/QSYS.LIB/RLPLIB.LIB/LDRW.OUTQ");
     PrintParameterList parms = new PrintParameterList();
     parms.setParameter(PrintObject.ATTR_PRTDEVTYPE, "*LINE");

     // инициализация формата записи для занесения данных
               RecordFormat recfmt = initializeRecordFormat();

     // создание записи и загрузка данных для печати...
               Record record = new Record(recfmt);
     createRecord(record);

     SpooledFileOutputStream os = null;
     try {
         // создание буферного файла вывода для хранения данных записи
                        os = new SpooledFileOutputStream(system_, parms, null, outQ);
     }
     catch (Exception e) {
         System.out.println("При создании буферного файла произошла ошибка");
         e.printStackTrace();
     }

     // создание программы для внесения записей строковых данных
     LineDataRecordWriter ldw;
     ldw = new LineDataRecordWriter(os, ccsid_, system_);

     // внесение записи
     ldw.writeRecord(record);

     // закрытие потока вывода
     os.close();
   }

   catch(Exception e)
   {
     failed(e, "Возникла исключительная ситуация.");
   }
```

LineDataRecordWriter Javadoc

Record Javadoc

RecordFormat Javadoc

Очереди данных

Классы DataQueue позволяют программам на Java работать с очередями данных сервера.

R System i5 data queues have the following characteristics:

- Очереди данных обеспечивают быструю передачу данных между заданиями. По этой причине их часто применяют для синхронизации заданий.

- С очередями данных могут одновременно работать несколько заданий.
- Формат сообщений очереди данных не фиксирован. В отличие от файлов баз данных, указывать поля не обязательно.
- Очереди данных могут применяться как при синхронной, так и при асинхронной обработке.
- Сообщения в очереди данных могут быть упорядочены одним из следующих способов:
  - ″Последним вошел - первым вышел″ (LIFO). Первым извлекается последнее (самое новое) сообщение.
  - ″Первым вошел - первым вышел″ (FIFO). Первым извлекается первое (самое старое) сообщение.
  - По ключу. С каждым сообщением в очереди данных связан определенный ключ. Для извлечения сообщения необходимо указать связанный с ним ключ.

Классы очередей данных предоставляют программе на Java полный набор интерфейсов для работы с очередями данных сервера. По этой причине очереди данных - это идеальное средство взаимодействия программ на Java с программами сервера, написанными на других языках.

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

При работе с классами очередей данных объект AS400 устанавливает соединение с сервером. Информация об управлении соединениями приведена в разделе управление соединениями.

Кроме того, с каждым объектом очереди данных связан полный путь к очереди данных в интегрированной файловой системе (IFS). Объектам очередей данных в IFS соответствует тип DTAQ. Дополнительная информация приведена в разделе Пути к объектам IFS.

Очереди данных с последовательным извлечением и извлечением по ключу

The data queue classes support sequential and keyed data queues.:

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

When data is read from a data queue, the data is placed in a DataQueueEntry object. В этом объекте могут храниться данные из очередей обоих типов. Additional data available when reading from a keyed data queue is placed in a KeyedDataQueueEntry object that extends the DataQueueEntry class.

Классы очередей данных не изменяют данные во время их чтения из очереди или записи в очередь. Форматирование данных должно выполняться программой на Java. Для этого предусмотрены классы преобразования данных.

Пример: Применение классов DataQueue и DataQueueEntry

В приведенном ниже примере создается объект DataQueue, из него считываются данные в объект DataQueueEntry, после чего соединение с системой разрывается.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
// Создание объекта AS400
AS400 sys = new AS400("mySystem.myCompany.com");

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

// Чтение данных из очереди
DataQueueEntry dqData = dq.read();
```

```
// Получение данных из объекта DataQueueEntry
byte[] data = dqData.getData();

// ... обработка данных

// После завершения работы с очередями данных - отсоединение
sys.disconnectService(AS400.DATAQUEUE);
```

Информация, связанная с данной

BaseDataQueue Javadoc

DataQueue Javadoc

KeyedDataQueue Javadoc

DataQueueEntry Javadoc

KeyedDataQueueEntry Javadoc

Очереди данных с последовательным извлечением:

Данные в очередях с последовательным извлечением могут быть упорядочены по принципу ″Первым вошел - первым вышел″ (FIFO) или ″Последним вошел - первым вышел″ (LIFO).

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

- Create a data queue on the server. Программа на Java должна указать максимальный размер передаваемой записи. Кроме того, при создании программа на Java может указать набор дополнительных параметров (FIFO или LIFO, следует ли сохранять информацию об отправителе, следует ли сохранять на диске, описание очереди).
- Peek at an entry on the data queue without removing it from the queue. При отсутствии данных выполнение программы на Java может быть приостановлено до поступления нужной записи или продолжено.
- Read an entry off the queue. При отсутствии данных выполнение программы на Java может быть приостановлено до поступления нужной записи или продолжено.
- Write an entry to the queue.
- Clear all entries from the queue.
- Delete the queue.

Класс BaseDataQueue содержит дополнительные методы для получения атрибутов очереди данных.

Примеры: Работа с последовательными очередями данных

В приведенных ниже примерах производитель помещает элементы в очередь данных, а потребитель извлекает их и обрабатывает:

“Пример: Применение классов DataQueue для занесения записей в очередь данных” на стр. 490

“Пример: Применение классов DataQueue для считывания записей из очереди данных” на стр. 493

BaseDataQueue Javadoc

DataQueue Javadoc

Очереди данных с извлечением по ключу:

Классы BaseDataQueue и KeyedDataQueue содержат следующие методы для работы с очередями данных с извлечением по ключу.

- Create a keyed data queue on the system. Программа на Java должна указать размер ключа и максимальный размер передаваемой записи. Кроме того, при создании программа на Java может указать набор дополнительных параметров (права доступа, следует ли сохранять информацию об отправителе, следует ли сохранять на диске, описание очереди).

- Peek at an entry based on the specified key without removing it from the queue. При отсутствии записи с указанным ключом выполнение программы на Java может быть приостановлено до поступления нужной записи или продолжено.
- Read an entry off the queue based on the specified key. При отсутствии записи с указанным ключом выполнение программы на Java может быть приостановлено до поступления нужной записи или продолжено.
- Write a keyed entry to the queue.
- Clear all entries or all entries that match a specified key.
- Delete the queue.

Классы BaseDataQueue и KeyedDataQueue содержат дополнительные методы для получения атрибутов очереди данных.

Примеры: Работа с ключевыми очередями данных

В приведенных ниже примерах производитель помещает элементы в очередь данных, а потребитель извлекает их и обрабатывает:

“Пример: Применение класса KeyedDataQueue” на стр. 499

“Пример: Применение классов KeyedDataQueue для считывания записей из очереди данных” на стр. 502

BaseDataQueue Javadoc

KeyedDataQueue Javadoc

Цифровые сертификаты

Цифровые сертификаты - это документы с цифровой подписью, используемые для защиты транзакций в Internet.

Цифровой сертификат необходим для установления защищенного соединения с помощью SSL.

Цифровой сертификат включает следующие элементы:

- Общий ключ шифрования пользователя
- Имя и адрес пользователя
- Цифровая подпись независимой сертификатной компании (CA). Подпись CA означает, что пользователь является уполномоченным лицом
- Дата создания сертификата
- Дата истечения срока действия сертификата

Будучи администратором защищенного сервера, вы можете поместить на сервер надежный базовый ключ сертификатной компании. Это означает, что сервер будет разрешать доступ всем пользователям, сертифицированным данной компанией.

Цифровые сертификаты поддерживают шифрование, обеспечивая защищенную передачу данных с использованием личного ключа.

R Для создания цифровых сертификатов можно использовать инструмент javakey. (Дополнительная
R информация о javakey и средствах защиты Java приведена на Web-странице Sun Microsystems, Inc., Java
R Security .) The IBM Toolbox for Java licensed program has classes that administer digital certificates on the
R system.

В классах AS400Certificate предусмотрены методы для работы с сертификатами в кодировке X.509 ASN.1. Классы позволяют выполнять следующие действия:

- Считывать и задавать информацию сертификата.
- Получать списки сертификатов для профайла или контрольного списка.
- Управлять сертификатами - например, добавлять сертификаты в пользовательский профайл или удалять сертификаты из контрольного списка.

При использовании класса сертификатов объект AS400 автоматически подключается к серверу. Информация об управлении соединениями приведена в разделе управление соединениями.

На сервере сертификат принадлежит контрольному списку или пользовательскому профайлу.
- The AS400CertificateUserProfileUtil class has methods for managing certificates on a user profile.
- Класс AS400CertificateVldlUtil содержит методы для управления сертификатами в контрольных списках.

Для работы с AS400CertificateUserProfileUtil и AS400CertificateVldlUtil необходимо установить базовый компонент 34 (Диспетчер цифровых сертификатов) операционной системы. Эти классы расширяют класс AS400CertificateUtil - абстрактный класс, содержащий методы, общие для обоих подклассов.

Класс AS400Certificate содержит методы для чтения и записи данных сертификата.Данные представлены в виде массива байт. Пакет Java.Security виртуальной машины Java 1.2 включает классы, обеспечивающие доступ к отдельным полям сертификата.

Получение списка сертификатов

Для получения списка сертификатов программа на Java должна выполнить следующие операции:
1. Создать объект AS400.
2. Создать соответствующий объект для сертификата. Для работы с сертификатами в пользовательских профайлах и в контрольных списках применяются разные классы (AS400CertificateUserProfileUtil и AS400CertificateVldlUtil, соответственно).
3. Создать критерий выбора на основе атрибутов сертификата. The AS400CertificateAttribute class contains attributes used as selection criteria. Критерий выбора определяется набором объектов, соответствующих необходимым атрибутам. Например, могут быть выбраны только те сертификаты, который относятся к определенному пользователю или организации.
4. Создать пользовательское пространство на сервере и поместить в него сертификат. При создании списка могут использоваться большие объемы данных. До получения сертификатов программой на Java они помещаются в пользовательское пространство. Use the listCertificates() method to put the certificates into the user space.
5. Use the AS400CertificateUtil getCertificates() method to retrieve certificates from the user space.

Пример: Просмотр цифровых сертификатов

Приведенный ниже пример позволяет получить список сертификатов из контрольного списка. В список включаются только сертификаты, принадлежащие определенному лицу.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
        // Создание объекта AS400.
Сертификаты находятся в этой системе.
     AS400 sys = new AS400("mySystem.myCompany.com");

        // Создание объекта для сертификатов.
     AS400CertificateVldlUtil certificateList =
        new AS400CertificateVldlUtil(sys, "/QSYS.LIB/MYLIB.LIB/CERTLIST.VLDL");

        // Создание списка атрибутов сертификатов. Необходимо создать сертификат
        // только для одного человека, поэтому в списке только один элемент.
     AS400CertificateAttribute[] attributeList = new AS400CertificateAttribute[1];
```

```
     attributeList[0] =
        new AS400CertificateAttribute(AS400CertificateAttribute.SUBJECT_COMMON_NAME, "Jane Doe");
// Получение списка, соответствующего критериям. Пользовательское пространство "myspace"
        // в библиотеке "mylib" будет применяться для хранения сертификатов.
        // Пользовательское пространство должно существовать на момент вызова API.
     int count = certificateList.listCertificates(attributeList, "/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");

        // Получение сертификатов из пользовательского пространства.
     AS400Certificates[] certificates =
        certificateList.getCertificates("/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC", 0, 8);
// Обработка сертификатов
```

AS400CertificateUserProfileUtil Javadoc

AS400CertificateVldUtil Javadoc

AS400CertificateAttribute Javadoc

Класс EnvironmentVariable

The IBM Toolbox for Java EnvironmentVariable class and the EnvironmentVariableList class enable you to access and set System i5 system-level environment variables.

У каждой переменной есть уникальный идентификатор, состоящий из имени системы и имени переменной среды. Каждая переменная среды связана с CCSID (по умолчанию - с CCSID текущего задания); CCSID описывает место хранения содержимого переменной.

Примечание: Переменные среды отличаются от системных значений, несмотря на то, что часто они используются в одинаковых целях. За дополнительной информацией о работе с системными значениями обратитесь к разделу Класс SystemValues.

Объект EnvironmentVariable позволяет выполнять над переменной среды следующие действия:

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

Пример: Создание, настройка и получение значений переменных среды

В следующем примере создаются две переменные среды, им присваиваются значения, после чего эти значения считываются.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

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

// Создание переменной среды, задающей цвет текста, и настройка красного цвета текста.
EnvironmentVariable fg = new EnvironmentVariable(system, "FOREGROUND");
fg.setValue("RED");

// Создание переменной среды, задающей цвет фона, и получение ее значения.
EnvironmentVariable bg = new EnvironmentVariable(system, "BACKGROUND");
String background = bg.getValue();
```

EnvironmentVariable Javadoc

EnvironmentVariableList Javadoc

Исключительные ситуации

The IBM Toolbox for Java access classes throw exceptions when device errors, physical limitations, programming errors, or user input errors occur. Исключительные ситуации разделены на классы в соответствии с характером ошибок, а не по месту их возникновения.

При возникновении большинства исключительных ситуаций передается следующая информация:

- **Тип ошибки:** Объект исключительной ситуации указывает тип ошибки. Ошибки одного типа объединены в класс исключительных ситуаций.
- **Сведения об ошибке:** Объект исключительной ситуации содержит код возврата, указывающий на причину возникновения ошибки. Коды возврата являются константами, определенными в классе исключительных ситуаций.
- **Текст ошибки:** Объект исключительной ситуации содержит строку - описание ошибки. Эта строка переведена на язык, определяемый локалью виртуальной машины Java клиента.

Пример: Обработка исключительной ситуации

В следующем примере перехватывается исключительная ситуация, считывается код возврата и выводится текст об ошибке:

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
// Все предварительные операции по удалению файла на сервере с помощью класса IFSFile
// выполнены. Попытайтесь удалить файл.
try
{
   aFile.delete();

}

// При удалении возникла ошибка.
catch (ExtendedIOException e)
{
   // Показать преобразованную строку, содержащую описание причины сбоя
   // при удалении.
   System.out.println(e);

   // Получение кода возврата объекта исключительной ситуации и просмотр
   // дополнительной информации для этого кода возврата.
   int rc = e.getReturnCode()

   switch (rc)
   {
      case ExtendedIOException.FILE_IN_USE:
         System.out.println("Удаление невозможно - файл используется"):
         break;

      case ExtendedIOException.PATH_NOT_FOUND:
         System.out.println("Удаление невозможно - путь не найден"):
         break;

      // Для каждой конкретной ошибки, которую необходимо отследить...

      default:
         System.out.println("Удаление невозможно - rc = ");
         System.out.println(rc);
   }
}
```

FileAttributes class

The IBM Toolbox for Java FileAttributes class represents the set of file attributes that can be retrieved and set through the Get Attributes (Qp0lGetAttr) and Set Attributes (Qp0lSetAttr) APIs.

To obtain the attributes of an object, the object must exist and the caller must have authority to it. Only attributes supported by the specific file system, object type, and system operating release can be retrieved or set.

You can find a complete list of the available attributes in the FileAttributes Javadoc.

Информация, связанная с данной

FileAttributes Javadoc

Get Attributes (Qp0lGetAttr) API

Set Attributes (Qp0lSetAttr) API

Класс FTP

The IBM Toolbox for Java FTP class provides a programmable interface to FTP functions.

Вам больше не требуется вызывать java.runtime.exec() или предлагать пользователям запустить команды FTP в отдельном приложении. Функции FTP теперь можно вызывать непосредственно из вашей программы. Класс FTP позволяет выполнять следующие операции:

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

Пример: Копирование файлов с сервера с помощью FTP

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

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

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

for (int i = 0; i < entries.length; i++)
{
  System.out.println("Копирование " + entries[i]);
  try
  {
    client.get(entries[i], "c:\\ftptest\\" + entries[i]);
  }
  catch (Exception e)
  {
    System.out.println("  скопировать данные не удалось, вероятная причина сбоя - ошибка каталога");
  }
}

client.disconnect();
```

FTP - это стандартный интерфейс, позволяющий работать с различными FTP-серверами. За соответствие запросов семантике сервера отвечает программист.

Подкласс AS400FTP

R While the FTP class is a generic FTP interface, the AS400FTP subclass is written specifically for the FTP server on the
R server. That is, it understands the semantics of the FTP server on the System i5. Например, этот класс содержит
R функции, применяемые при передаче файла сохранения на сервер, что позволяет выполнять эту операцию
R автоматически. AS400FTP also ties into the security facilities of the IBM Toolbox for Java. As with other IBM
R Toolbox for Java classes, AS400FTP depends on the AS400 object for system name, user ID, and password.

Пример: Сохранение файла на сервере с помощью подкласса AS400FTP

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

В приведенном ниже примере файл сохранения передается на сервер. Обратите внимание на то, что приложение не устанавливает двоичный режим передачи и не вызывает функцию Toolbox CommandCall для создания файла сохранения. Так как указано расширение .savf, класс AS400FTP распознает тип файла и выполняет операцию сохранения автоматически.

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

FTP Javadoc

AS400FTP Javadoc

Классы IFS

The integrated file system classes allow a Java program to access files in the System i5integrated file system of an as a stream of bytes or a stream of characters. The integrated file system classes were created because the java.io package does not provide file redirection and other System i5 functionality.

Набор функций, предоставляемых классами IFSFile, содержит набор функций классов ввода-вывода из пакета java.io в качестве подмножества. Все методы классов FileInputStream, FileOutputStream и RandomAccessFile из пакета java.io поддерживаются классами интегрированной файловой системы.

Кроме описанных выше методов, эти классы включают методы, позволяющие выполнять следующие задачи:

- Запрещать доступ к файлу во время его использования
- Разрешать открытие, создание или замену файла
- Блокировать часть файла и запретить доступ к этой части во время использования файла
- Более эффективно считывать содержимое каталога
- Заносить в кэш содержимое каталога для повышения производительности за счет сокращения числа запросов к серверу
- Определять объем свободного пространства в файловой системе сервера
- Предоставлять аплетам Java доступ к файлам сервера
- Считывать и записывать информацию в виде текста, а не в виде двоичных данных
- Определять тип объекта файловой системы QSYS.LIB (логический файл, физический файл, файл сохранения и так далее)

Through the integrated file system classes, the Java program can directly access stream files on the system. Программа на Java может по-прежнему использовать пакет java.io, однако в этом случае клиентская операционная система должна поддерживать метод перенаправления. For example, if the Java program is
L running on a Windows 95 or Windows NT operating system, the Network Drives function of System i Access for
L Windows is required to redirect java.io calls to the system. With the integrated file system classes, you do not need
L System i Access for Windows.

A required parameter of the integrated file system classes is the AS400 object that represents the system that contains the file. Using the integrated file system classes causes the AS400 object to connect to the system. Информация об управлении соединениями приведена в разделе управление соединениями.

Для работы классов интегрированной файловой системы необходимо указывать иерархическое имя объекта интегрированной файловой системы. При указании пути следует применять косую черту. Например, путь доступа к файлу FILE1 в каталоге DIR1/DIR2 выглядит следующим образом:

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

R **Примеры: Применение классов IFS**

R “Пример: Применение классов IFS для копирования файла из одного каталога в другой” на стр. 510 shows
R how to use the integrated file system classes to copy a file from one directory to another on the system.

R “Пример: Применение классов IFS для просмотра содержимого каталога” на стр. 512 shows how to use the
R integrated file system classes to list the contents of a directory on the system.

Класс IFSFile:

The IBM Toolbox for Java IFSFile class represents an object in the System i integrated file system.

Методы IFSFile соответствуют операциям, выполняемым над всем объектом. Для чтения и записи в файл применяются классы IFSFileInputStream, IFSFileOutputStream и IFSRandomAccessFile. Класс IFSFile позволяет программе на Java выполнять следующие операции:

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

Список файлов каталога можно получить с помощью метода list() или listFiles():

- При первом вызове метода listFiles() информация обо всех файлах заносится в кэш. В результате последующие запросы на получение атрибутов файлов обрабатываются быстрее, поскольку необходимая информация берется из кэша. Например, при вызове метода isDirectory() для объекта IFSFile, возвращенного методом listFiles(), не потребуется отправлять запрос серверу.
- Метод list() получает информацию о каждом файле с помощью отдельного запроса, поэтому этот метод работает медленнее и сильнее зависит от производительности сервера.

Примечание: При применении метода listFiles() информация в кэше может устареть, поэтому может понадобиться ее обновление с помощью метода listFiles().

Примеры

Ниже приведен пример применения класса IFSFile:

- “Пример: Создание каталога” на стр. 505
- “Пример: Применение исключительных ситуаций IFSFile для отслеживания ошибок” на стр. 506
- “Пример: Просмотр файлов с расширением .txt” на стр. 507
- “Пример: Применение метода listFiles() класса IFSFile для просмотра содержимого каталога” на стр. 507

  Информация, связанная с данной

  IFSFile Javadoc

Класс IFSJavaFile:

This IBM Toolbox for Java class represents a file in the System i5 integrated file system and extends the java.io.File class. IFSJavaFile allows you to write files for the java.io.File interface that access integrated file systems.

IFSJavaFile позволяет создавать переносимые интерфейсы, совместимые с java.io.File, и использует в точности те же ошибки и исключительные ситуации, что и java.io.File. В классе IFSJavaFile применяются функции диспетчера защиты из java.io.File, однако, в отличие от java.io.File, IFSJavaFile поддерживает функции защиты всегда.

Класс IFSJavaFile применяется совместно с IFSFileInputStream и IFSFileOutputStream. Он не поддерживает java.io.FileInputStream и java.io.FileOutputStream.

IFSJavaFile основан на IFSFile, однако его интерфейс ближе к java.io.File, чем интерфейс IFSFile. IFSFile является альтернативой классу IFSJavaFile.

Список файлов каталога можно получить с помощью метода list() или listFiles():

- При первом вызове метода listFiles() информация обо всех файлах заносится в кэш. After that, information about each file is retrieved from the cache.
- The list() method retrieves information about each file in a separate request, making it slower and more demanding of server resources.

Примечание: При применении метода listFiles() информация в кэше может устареть, поэтому может понадобиться ее обновление.

Пример: Применение класса IFSJavaFile

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведен пример использования класса IFSJavaFile:

```
// Работа с файлом /Dir/File.txt в системе flash.
AS400 as400 = new AS400("flash");
IFSJavaFile file = new IFSJavaFile(as400, "/Dir/File.txt");

// Определение каталога, в котором расположен файл
String directory = file.getParent();

// Определение имени файла
String name = file.getName();

// Определение размера файла
long length = file.length();

// Определение даты последнего изменения файла
Date date = new Date(file.lastModified());

// Удаление файла
if (file.delete() == false)
{
  // Вывод сообщения об ошибке
  System.err.println("Удаление файла невозможно.");
}

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 ("Исключительная ситуация: " + e.getMessage());
}
```

Информация, связанная с данной

IFSJavaFile Javadoc

IFSFileInputStream:

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

Как и в классе IFSFile, в IFSFileInputStream присутствуют методы, дублирующие методы класса FileInputStream из пакета java.io. In addition to these methods, IFSFileInputStream has additional methods specific to the System i5 platform. Класс IFSFileInputStream позволяет программе на Java выполнять следующие операции:

- Open a file for reading. Файл должен существовать, так как этот класс не создает файлов на сервере. В конструкторе класса можно задать режим использования файла.
- Determine the number of bytes in the stream.
- Read bytes from the stream.
- Skip bytes in the stream.
- Lock or unlock bytes in the stream.
- Закрыть файл.

Как и класс FileInputStream из пакета java.io, этот класс позволяет программе на Java считывать поток байт из файла. Программа Java читает байты последовательно и может пропускать байты при чтении.

Кроме методов класса FileInputStream, IFSFileInputStream предоставляет программе на Java возможность выполнять следующие операции:

- Блокировать и разблокировать байты в потоке. Дополнительная информация приведена в разделе IFSKey.
- Устанавливать режим использования файла при его открытии. Дополнительная информация приведена в разделе Режимы совместного использования.

Пример: Применение класса IFSFileInputStream

Ниже приведен пример использования класса IFSFileInputStream:

```
                    // Создание объекта AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание файлового объекта,
                    // соответствующего файлу
IFSFileInputStream aFile = new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");

                    // Определение числа байт
                    // в файле
int available = aFile.available();

                    // Выделение памяти под буфер для хранения данных
byte[] data = new byte[10240];

                    // Считывание файла блоками по 10 Кб
for (int i = 0; i < available; i += 10240)
{
    aFile.read(data);
```

```
}

                    // Закрытие файла
aFile.close();
```

Информация, связанная с данной

IFSFileInputStream Javadoc

Класс IFSTextFileInputStream:

IFSTextFileInputStream устарел и заменен классом IFSFileReader.

Класс IFSTextFileInputStream представляет поток символьных данных, считываемых из файла. Данные, считанные из объекта IFSTextFileInputStream, передаются программе на Java в виде объекта String Java, поэтому они всегда представлены в формате Unicode. При открытии файла объект IFSTextFileInputStream определяет CCSID данных, содержащихся в файле. Если данные представлены в кодировке, отличной от Unicode, объект IFSTextFileInputStream преобразует их в Unicode перед тем, как вернуть вызывающей программе на Java. Если преобразовать данные невозможно, вызывается исключительная ситуация UnsupportedEncodingException.

Ниже приведен пример использования класса IFSTextFileInputStream.

```
                    // Работа с файлом /File в системе
                    // mySystem.
AS400 as400 = new AS400("mySystem");
IFSTextFileInputStream file = new IFSTextFileInputStream(as400, "/File");

                    // Чтение первых 4 байт
                    // в файле.
String s = file.read(4);

                    // Вывод прочитанных символов.                    // При необходимости данные
                    // могут открывать этот файл.
// При необходимости данные будут преобразованы в
                    // Unicode объектом
                    // IFSTextFileInputStream.
System.out.println(s);

                    // Закрытие файла
file.close();
```

Ссылки, связанные с данной

“IFSFileReader”
Применяется для чтения символьных файлов интегрированной файловой системы.

IFSFileReader:

Применяется для чтения символьных файлов интегрированной файловой системы.

IFSFileReader применяется для чтения потоков символов. IFSFileReader заменяет IFSTextFileOutputStream.

Пример: Применение IFSFileReader

Ниже приведен пример применения класса IFSFileReader:

```
import java.io.BufferedReader;

 // Работа с файлом /File1 в системе eniac.
AS400 system = new AS400("eniac");
IFSFile file = new IFSFile(system, "/File1");
BufferedReader reader = new BufferedReader(new IFSFileReader(file));

 // Прочитать первую строку файла, преобразовать символы.
String line1 = reader.readLine();
```

```
 // Вывод прочитанной строки.
System.out.println(line1);

 // Закрыть объект.
reader.close();
```

Информация, связанная с данной

IFSFileReader Javadoc

Класс IFSFileOutputStream:

Класс IFSFileOutputStream представляет поток вывода для записи данных в файл сервера.

Как и в классе IFSFile, в IFSFileOutputStream присутствуют методы, дублирующие методы класса FileOutputStream из пакета java.io. Помимо этого в классе IFSFileOutputStream предусмотрены методы, относящиеся непосредственно к серверу. Класс IFSFileOutputStream позволяет программе на Java выполнять следующие операции:

- Open a file for writing. Если файл существует, он заменяется. С помощью конструкторов класса можно задать режим использования файла и указать, нужно ли сохранять содержимое существующего файла.
- Write bytes to the stream.
- Commit to disk the bytes that are written to the stream.
- Lock or unlock bytes in the stream.
- Закрыть файл.

Как и класс FileOutputStream из пакета java.io, этот класс позволяет программе на Java последовательно записывать поток байт в файл.

Кроме методов класса FileOutputStream, IFSFileOutputStream предоставляет программе на Java возможность выполнять следующие операции:

- Блокировать и разблокировать байты в потоке. Дополнительная информация приведена в разделе IFSKey.
- Устанавливать режим использования файла при его открытии. Дополнительная информация приведена в разделе Режимы совместного использования.

Пример: Применение класса IFSFileOutputStream

Ниже приведен пример использования класса IFSFileOutputStream:

```
                    // Создание объекта AS400
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание файлового объекта,
                    // соответствующего файлу
    IFSFileOutputStream aFile = new IFSFileOutputStream(sys,"/mydir1/mydir2/myfile");

                    // Запись в файл
    byte i = 123;
    aFile.write(i);

                    // Закрытие файла
    aFile.close();
```

IFSFileOutputStream Javadoc

Класс IFSTextFileOutputStream:

IFSTextFileOutputStream устарел и заменен классом IFSFileWriter.

Класс IFSTextFileOutputStream представляет поток символьных данных, записываемых в файл. Объекту IFSTextFileOutputStream передаются данные в виде объекта String Java, поэтому записываемые данные всегда представлены в кодировке Unicode. При этом объект IFSTextFileOutputStream может при записи перекодировать данные в требуемый CCSID. По умолчанию в файл записываются символы Unicode, однако программа на Java может задать целевой CCSID перед открытием файла. В этом случае объект IFSTextFileOutputStream преобразует символы из кодировки Unicode в кодировку с указанным CCSID перед записью данных в файл. Если преобразовать данные невозможно, вызывается исключительная ситуация UnsupportedEncodingException.

Пример: Применение класса IFSTextFileOutputStream

Ниже приведен пример использования класса IFSTextFileOutputStream.

```
                      // Работа с файлом /File в системе
                      // mySystem.
     AS400 as400 = new AS400("mySystem");
     IFSTextFileOutputStream file = new IFSTextFileOutputStream(as400, "/File");

                      // Запись строки (объекта String) в файл.
                      // Так как CCSID не был задан
                      // перед записью в файл,
                      // данные будут записаны
                      // в кодировке Unicode.
// Файл будет отмечен как
                      // файл данных Unicode.
     file.write("Добрый день");

                      // Закрытие файла
     file.close();
```

Ссылки, связанные с данной

"IFSFileWriter"
IFSFileWriter применяется для записи символьных файлов интегрированной файловой системы. IFSFileWriter применяется для записи потоков символов.

IFSFileWriter:

IFSFileWriter применяется для записи символьных файлов интегрированной файловой системы. IFSFileWriter применяется для записи потоков символов.

IFSFileWriter заменяет IFSTextFileOutputStream.

Пример: Применение IFSFileWriter

Ниже приведен пример применения класса IFSFileWriter:

```
import java.io.PrintWriter;
import java.io.BufferedWriter;
 // Работа с файлом /File1 в системе mysystem.
AS400 as400 = new AS400("mysystem");
IFSFile file = new IFSFile(system, "/File1");
PrintWriter writer = new PrintWriter(new BufferedWriter(new IFSFileWriter(file)));
 // Записать строку в файл, преобразовать символы.
writer.println(text);
 // Закрытие файла.
writer.close();
```

Информация, связанная с данной

IFSFileWriter Javadoc

IFSRandomAccessFile:

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

Программа на Java может считывать и записывать данные в файл последовательно или в произвольном порядке. Как и в классе IFSFile, в IFSRandomAccessFile существуют методы, дублирующие методы RandomAccessFile из пакета java.io. In addition to these methods, IFSRandomAccessFile has additional methods specific to System i5. С помощью класса IFSRandomAccessFile программа на Java может выполнять следующие действия:

- Open a file for read, write, or read/write access. При открытии программа на Java может задать режим использования файла и опцию открытия/создания файла.
- Read data at the current offset from the file.
- Write data at the current offset to the file.
- Get or set the current offset of the file.
- Закрыть файл.

Кроме методов класса RandomAccessFile из пакета java.io, класс IFSRandomAccessFile предоставляет программе на Java возможность выполнять следующие операции:

- Committing to disk bytes written.
- Locking or unlocking bytes in the file.
- Блокировать и разблокировать байты в потоке. Дополнительная информация приведена в разделе IFSKey.
- Устанавливать режим использования файла при его открытии. Дополнительная информация приведена в разделе Режимы совместного использования.
- Устанавливать опцию открытия/создания файла. Программа на Java может задать один из следующих вариантов:
  - Открыть файл, если он существует, в противном случае - создать файл.
  - Заменить файл, если он существует, в противном случае - создать файл.
  - Не открывать файл, если он существует, в противном случае - создать файл.
  - Открыть файл, если он существует, в противном случае - не открывать файл.
  - Заменить файл, если он существует, в противном случае - не открывать файл.

Пример: Применение класса IFSRandomAccessFile

Ниже приведен пример использования класса IFSRandomAccessFile для записи данных в файл с интервалом в 1024 байта.

```
                    // Создание объекта AS400.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание файлового объекта,
                    // соответствующего файлу.
    IFSRandomAccessFile aFile = new IFSRandomAccessFile(sys,"/mydir1/myfile", "rw");

                    // Указание данных для записи.
    byte i = 123;

                    // Запись данных в файл, выполняемая
                    // 10 раз с интервалом в 1024 байта.
    for (int j=0; j<10; j++)
    {
                    // Изменение текущего смещения.
       aFile.seek(j * 1024);

                    // Запись данных в файл. Текущее
// смещение увеличится на размер
                    // записанных данных.
       aFile.write(i);
    }

                    // Закрытие файла
    aFile.close();
```

Информация, связанная с данной

IFSRandomAccessFile Javadoc

IFSFileDialog:

The IBM Toolbox for Java IFSFileDialog class allows you to traverse the file system and select a file.

R This class uses the IFSFile class to traverse the list of directories and files in the System i5 integrated file system.
R Методы класса позволяют программам на Java задавать текст кнопок в окне диалога и устанавливать
R фильтры. Обратите внимание, что существует версия IFSFileDialog, использующая Swing 1.1.

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

Пример: Применение IFSFileDialog

В следующем примере показано, как создать окно диалога с двумя фильтрами и определить текст на его кнопках.

```
                    // Создание объекта AS400.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта окна диалога путем
                    // указания текста строки заголовка окна
                    // и целевого сервера.
IFSFileDialog dialog = new IFSFileDialog(this, "Текст заголовка", sys);

                    // Создание списка фильтров, затем
                    // the filters in the dialog.                        // Первый фильтр будет установлен
                    // при первом выводе окна диалога.
FileFilter[] filterList = {new FileFilter("Все файлы (*.*)", "*.*"),
                           new FileFilter("Файлы HTML (*.HTML)", "*.HTM")};

dialog.setFileFilter(filterList, 0);

                    // Указание текста кнопок
                    // окна диалога.
dialog.setOkButtonText("Открыть");
dialog.setCancelButtonText("Отмена");

                    // Выдается окно диалога. Если пользователь выбрал
                    // файл с помощью кнопки Открыть,
                    // то программа считывает и выводит
                    // имя выбранного файла.
if (dialog.showDialog() == IFSFileDialog.OK)
   System.out.println(dialog.getAbsolutePath());
```

Информация, связанная с данной

FileFilter Javadoc

IFSFileDialog Javadoc

Класс IFSKey:

Когда программа на Java работает с файлом, доступным другим программам Java, она может заблокировать байтовый фрагмент в этом файле на определенное время. В течение этого времени программа будет обладать исключительным доступом к фрагменту. When a lock is successful, the integrated file system classes return an IFSKey object.

Этот объект передается методу unlock() для того, чтобы указать, какие байты следует разблокировать. При закрытии файла система снимает все блокировки, установленные для файла (т.е. все блокировки, не снятые программой).

Пример: Применение класса IFSKey

Ниже приведен пример использования класса IFSKey:

```
                    // Создание объекта AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Open an input stream.                        // вызывается в режиме share_all,
                    // поэтому другие программы также
                    // могут открывать этот файл.
     IFSFileInputStream aFile =
                  new IFSFileInputStream(sys,"/mydir1/mydir2/myfile");

                    // Заблокировать первые 1024 байта файла.
                    // могут открывать этот файл.
Другие экземпляры
                    // не смогут считывать эти байты.
     IFSKey key = aFile.lock(1024);

                    // Прочитать первые 1024 байта из файла.
     byte data[] = new byte[1024];
     aFile.read(data);

                    // Разблокировать фрагмент файла.
     aFile.unlock(key);

                    // Закрытие файла
     aFile.close();
```

Информация, связанная с данной

IFSKey Javadoc

Режимы использования файлов:

При открытии файла программа на Java может задать режим его использования. Программа может выбрать исключительный доступ к файлу или разрешить параллельное использование файла другими программами.

Ниже приведен пример установки режима использования файла:

```
                    // Создание объекта AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание файлового объекта,
                    // соответствующего файлу. Поскольку в программе
                    // указан режим share-none, попытки
                    // открыть файл из других программ
                    // будут отклоняться до закрытия этой программы.
     IFSFileOutputStream aFile = new IFSFileOutputStream(sys,
                                                         "/mydir1/mydir2/myfile",
                                                         IFSFileOutputStream.SHARE_NONE,
                                                         false);

                    // Выполнение операций над файлом.

                    // Закрытие файла. Теперь другие
                    // программы могут его использовать.
     aFile.close();
```

1IFSSystemView:

IFSSystemView provides a gateway to the System i integrated file system, for use when constructing javax.swing.JFileChooser objects.

JFileChooser - это стандартный инструмент Java для создания окон диалога, позволяющих искать и выбирать файлы.

Пример: Применение IFSSystemView

Ниже приведен пример применения класса IFSSystemView:

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

 // Работа с каталогом /Dir в системе myAS400.
AS400 system = new AS400("myAS400");
IFSJavaFile dir = new IFSJavaFile(system, "/Dir");
JFileChooser chooser = new JFileChooser(dir, new IFSSystemView(system));
Frame parent = new Frame();
int returnVal = chooser.showOpenDialog(parent);
if (returnVal == JFileChooser.APPROVE_OPTION) {
IFSJavaFile chosenFile = (IFSJavaFile)(chooser.getSelectedFile());
  System.out.println("Вы выбрали файл " +
                   chosenFile.getName());
 }
```

Информация, связанная с данной

IFSSystemView Javadoc

ISeriesNetServer

Класс ISeriesNetServer представляет службу NetServer на сервере. Этот класс применяется для просмотра и изменения информации о состоянии и конфигурации службы NetServer.

ISeriesNetServer заменяет класс NetServer.

Информация, связанная с данной

ISeriesNetServer Javadoc

JavaApplicationCall

Класс JavaApplicationCall позволяет клиенту запускать программы на Java, расположенные на сервере, с помощью сервера JVM.

После подключения клиента к серверу класс JavaApplicationCall позволяет выполнить следующие действия:

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

Класс JavaApplicationCall - это класс, вызываемый из программы на Java. Однако в IBM IBM Toolbox for Java предусмотрены утилиты для вызова программ на Java, расположенных на сервере. Эти утилиты являются логически законченными программами на Java и могут быть запущены на рабочей станции. Дополнительная информация приведена в разделе Класс RunJavaApplication.

Пример

В этом примере в справочной документации Java по JavaApplicationCall описано, как можно запустить на сервере программу клиента, которая выводит фразу Hello, World!

JavaApplicationCall

Информация, связанная с данной

JavaApplicationCall Javadoc

Классы JDBC

JDBC - это интерфейс прикладных программ (API), который входит в пакет Java и позволяет программам на Java работать с широким спектром баз данных.

Поддерживаемые интерфейсы

В приведенной ниже таблице указаны поддерживаемые интерфейсы JDBC и API, необходимые для их применения:

+ **Примечание:** In V6R1, support for JDBC 4.0 is available only in the JTOpen version of IBM Toolbox for Java, not in
+ 5761-JC1. Hence, some of these interfaces may not be fully supported on your system.

Поддерживаемый интерфейс JDBC	**Необходимый API**
Blob provides access to binary large objects (BLOBs).	JDBC 2.1 core, enhanced support in JDBC 4.0
CallableStatement runs SQL stored procedures.	JDK 1.1, enhanced support in JDBC 4.0
Clob provides access to character large objects (CLOBs).	JDBC 2.1 core, enhanced support in JDBC 4.0
Connection represents a connection to a specific database.	JDK 1.1, enhanced support in JDBC 4.0
ConnectionPool represents a pool of Connection objects.	Дополнительный пакет JDBC 2.0
ConnectionPoolDataSource represents a factory for pooled AS400JDBCPooledConnection objects.	Дополнительный пакет JDBC 2.0
DatabaseMetaData provides information about the database as a whole.	JDK 1.1, enhanced support in JDBC 4.0
DataSource represents a factory for database connections.	JDBC 2.0 Optional Package, enhanced support in JDBC 4.0
Driver creates the connection and returns information about the driver version.	JDK 1.1
ParameterMetaData provides the ability to get information about the types and properties of the parameters in a PreparedStatement object.	JDBC 3.0 API, enhanced support in JDBC 4.0
+ PooledConnection provides hooks for connection pool + management.	JDBC 2.0 Optional Package, enhanced support in JDBC 4.0
PreparedStatement runs compiled SQL statements.	JDK 1.1, enhanced support in JDBC 4.0
ResultSet provides access to a table of data that is generated by running a SQL query or DatabaseMetaData catalog method.	JDK 1.1, enhanced support in JDBC 4.0
ResultSetMetaData provides information about a specific ResultSet.	JDK 1.1, enhanced support in JDBC 4.0
+ RowId represents an SQL ROWID value.	JDBC 4.0
RowSet is a connected row set that encapsulates a ResultSet.	Дополнительный пакет JDBC 2.0
Savepoint provides finer grained control within transactions	API JDBC 3.0
Statement runs SQL statements and obtains the results.	JDK 1.1, enhanced support in JDBC 4.0

Поддерживаемый интерфейс JDBC	Необходимый API
+ + + + + StatementEvent is sent to all StatementEventListeners which were registered with a PooledConnection. This occurs when the driver determines that a PreparedStatement that is associated with the PooledConnection has been closed or is determined by the driver to be invalid.	
+ + StatementEventListener registers to be notified of events that occur on PreparedStatements that are in the Statement pool.	JDBC 4.0
XAConnection is a database connection which participates in global XA transactions.	Дополнительный пакет JDBC 2.0
XAResource is resource manager for use in XA transactions.	Дополнительный пакет JDBC 2.0

Ссылки, связанные с данной

“JDBC” на стр. 319
JDBC - это интерфейс прикладных программ (API), который входит в пакет Java и позволяет программам на Java работать с широким спектром баз данных.

“Пример: Применение класса JDBCPopulate для создания и заполнения таблицы” на стр. 515
This program uses the IBM Toolbox for Java JDBC driver to create and populate a table.

“Пример: Применение класса JDBCQuery для отправки запроса к таблице” на стр. 526
This program uses the IBM Toolbox for Java JDBC driver to query a table and output its contents.

Информация, связанная с данной

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

AS400JDBCConnectionPoolDataSource Javadoc

AS400JDBCPooledConnection Javadoc

RowID

Класс AS400JDBCBlob:

Объект AS400JDBCBlob предназначен для работы с большими двоичными объектами (BLOB), например, звуковыми файлами (.wav) и файлами изображений (.gif).

Основное различие между классами AS400JDBCBlob и AS400JDBCBlobLocator заключается в типе сохраняемой информации об объекте blob. При работе с классом AS400JDBCBlob в базе данных сохраняется сам объект blob, что увеличивает размер файла базы данных. При работе с классом AS400JDBCBlobLocator в базе данных сохраняется только указатель на объект blob.

В классе AS400JDBCBlob предусмотрено свойство, задающее пороговый размер большого объекта. Большой объект (LOB), размер которого меньше порога, можно выбрать из базы данных целиком. Если размер LOB больше указанного порога, он извлекается по частям, что увеличивает длительность обмена данными с сервером. Чем больше пороговый размер LOB, тем меньше число обращений к серверу, но тем больше объем загружаемых данных (возможно, ненужных). При низком пороговом размере LOB число обращений к серверу увеличивается, однако вы будете получать только те данные LOB, которые действительно нужны. Информация о дополнительных свойствах приведена в разделе Свойства JDBC.

Класс AS400JDBCBlob позволяет выполнять следующие операции:

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

- Truncate the blob

Примеры

Ниже приведены примеры чтения из объекта blob и обновления объекта blob с помощью класса AS400JDBCBlob:

Пример: Чтение из объекта blob с помощью класса AS400JDBCBlob

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

Пример: Обновление объекта blob с помощью класса AS400JDBCBlob

```
     ResultSet rs =
statement.executeQuery ("Выберите объект BLOB в таблице");
     rs.absolute(5);
     Blob blob = rs.getBlob(1);
          // Изменение байтов в объекте blob, начиная с седьмого байта
          // объекта
     blob.setBytes (7, new byte[] { (byte) 57, (byte) 58, (byte) 98});
          //Обновление объекта blob в наборе результатов путем изменения blob,
          // начиная с седьмого байта и усечение объекта blob
          // после обновленных байтов (объект blob теперь содержит 9 байт).
     rs.updateBlob(1, blob);
          // Обновление базы данных.  При этом объект blob, хранящийся
// в базе данных, будет изменен, начиная с седьмого байта и
          // усечен после обновленных байтов.
     rs.updateRow ();
     rs.close();
```

Класс AS400JDBCBlobLocator

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

Класс AS400JDBCBlobLocator позволяет выполнять следующие операции:

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

  AS400JDBCBlob Javadoc

  AS400JDBCBlobLocator Javadoc

Интерфейс CallableStatement:

Объект CallableStatement предназначен для вызова хранимых процедур SQL. Вызываемая хранимая процедура должна уже содержаться в базе данных. Объект CallableStatement не содержит хранимую процедуру, а только вызывает ее.

Хранимая процедура вызывается с параметрами IN, OUT и INOUT. Она возвращает один или несколько объектов ResultSet. Для создания объекта CallableStatement предназначен метод Connection.prepareCall().

Объект CallableStatement позволяет объединить несколько команд SQL в одну группу и передать их в базу данных как один объект с помощью поддержки пакетного режима. Выполнение группы операторов в

пакетном режиме обычно занимает меньше времени, чем выполнение операторов по отдельности. Дополнительная информация о поддержке пакетного режима приведена в разделе Изменения, внесенные в поддержку JDBC.

CallableStatement позволяет получать и задавать параметры и столбцы по именам, несмотря на то, что применение индекса столбцов дает более высокую производительность.

Пример: Применение интерфейса CallableStatement

Ниже приведен пример работы с интерфейсом CallableStatement.

```
                    // Подключение к серверу.
     Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Создание объекта CallableStatement.
                    // Он выполняет предварительный
                    // вызов процедуры. Вместо
                    // вопросительных знаков будут
// подставлены входные
                    // и выходные параметры.
                    // Первые два параметра
                    // являются входными,
                    // а третий - выходным.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

                    // Настройка входных параметров.
     cs.setInt (1, 123);
     cs.setInt (2, 234);

                    // Регистрация типа выходного
                    // параметра.
     cs.registerOutParameter (3, Types.INTEGER);

                    // Запуск хранимой процедуры.
     cs.execute ();

                    // Получение значения выходного
                    // параметра.
     int sum = cs.getInt (3);

                    // Закрытие объектов CallableStatement и
                    // Connection.
     cs.close();
     c.close();
```

AS400JDBCCallableStatement Javadoc

Класс AS400JDBCClob:

Объект AS400JDBCClob предназначен для работы с большими символьными объектами (CLOB), например, большими документами.

Основное различие между классами AS400JDBCClob и AS400JDBCClobLocator заключается в типе сохраняемой информации об объекте clob. При работе с классом AS400JDBCClob в базе данных сохраняется сам объект clob, что увеличивает размер файла базы данных. При работе с классом AS400JDBCClobLocator в базе данных сохраняется только указатель на объект clob.

В классе AS400JDBCClob предусмотрено свойство, задающее пороговый размер большого объекта. Большой объект (LOB), размер которого меньше порога, можно выбрать из базы данных целиком. Если размер LOB больше указанного порога, он извлекается по частям, что увеличивает длительность обмена данными с сервером. Чем больше пороговый размер LOB, тем меньше число обращений к серверу, но тем больше объем загружаемых данных (возможно, ненужных). При низком пороговом размере LOB число

обращений к серверу увеличивается, однако вы будете получать только те данные LOB, которые действительно нужны. See “IBM Toolbox for Java JDBC properties” на стр. 328 for information about additional properties that are available.

Класс AS400JDBCClob позволяет выполнять следующие операции:

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

Примеры

Ниже приведены примеры чтения и обновления объекта clob с помощью класса AS400JDBCClob:

Пример: Чтение из объекта blob с помощью класса AS400JDBCBlob

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

Пример: Обновление объекта blob с помощью класса AS400JDBCBlob

```
ResultSet rs = statement.executeQuery ("Выберите объект CLOB в таблице");
rs.absolute(4);
Clob clob = rs.getClob (1);

     // Изменение символов в объекте clob, начиная с третьего
     // символа
clob.setString (3, "Small");

     // Обновление объекта clob в наборе результатов, начиная с  третьего
     // символа; усечение объекта clob в конце новой строки
     // (после этого объект clob будет содержать 7 символов).
rs.updateClob(1, clob);

     // Обновление базы данных. При этом в базе данных изменяется
// объект clob, начиная с третьего символа, а затем его
     // содержимое усекается по размеру новой строки.
rs.updateRow ();
rs.close();
```

Класс AS400JDBCClobLocator

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

Класс AS400JDBCClobLocator позволяет выполнять следующие операции:

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

AS400JDBCClob Javadoc

AS400JDBCClobLocator Javadoc

Класс AS400JDBCConnection:

The AS400JDBCConnection class provides a JDBC connection to a specific DB2 for i5/OS database.

Для создания объектов AS400JDBCConnection предназначен метод DriverManager.getConnection(). Дополнительная информация приведена в разделе “Регистрация драйвера JDBC” на стр. 79.

При создании соединения можно задать различные дополнительные свойства. Их можно указать в составе URL или в объекте java.util.Properties. В разделе “IBM Toolbox for Java JDBC properties” на стр. 328 приведен полный список свойств, поддерживаемых классом AS400JDBCDriver.

Примечание: Соединение может содержать до 9999 открытых операторов.

Класс AS400JDBCConnection позволяет работать с точками сохранения и поддерживает блокировку на уровне операторов. Кроме того, он частично поддерживает автоматически создаваемые ключи. Дополнительная информация об этих и других расширениях приведена в разделе “Расширенная поддержка JDBC в версии 5 выпуска 3” на стр. 324.

Если вы планируете применять паспорт Kerberos, укажите в объекте URL JDBC только имя системы (без пароля). Имя пользователя будет получено с помощью Общих служб защиты Java (JGSS), поэтому его не нужно указывать в URL JDBC. В объекте AS400JDBCConnection можно задать только один способ идентификации. Если вы зададите пароль, то все паспорта и разрешения Kerberos будут удалены. For more information, see the “Класс AS400” на стр. 21 and J2SDK, v1.4 Security Documentation .

Класс AS400JDBCConnection позволяет выполнять следующие операции:

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

Если применяется JDBC 3.0 и объект AS400JDBCConnection применяется для подключения к серверу с операционной системой i5/OS выпуска V5R2 или выше, то с его помощью можно выполнять следующие действия:

- Create a statement with a specific result set holdability (Statement, PreparedStatement, or CallableStatement object)
- Create a prepared statement that returns any auto-generated keys (when getGeneratedKeys() is called on the Statement object)
- Применять точки сохранения, предоставляющие гибкие средства управления транзакциями:
  - Set savepoints
  - Rollback savepoints
  - Release savepoints

AS400JDBCConnection Javadoc

AS400JDBCConnectionPool:

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

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

Источник данных пула соединений нельзя изменить после того, как был сделан запрос на соединение и пул был занят системой. To reset the connection pool data source, first call close() on the pool.

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

Примечание: Если соединения не возвращаются в пул, то размер пула растет, а соединения не используются повторно.

Set properties on the pool by using methods inherited from ConnectionPool. Ниже перечислены некоторые из этих свойств:

- максимальное число соединений в пуле
- максимальный срок действия соединения
- максимальное время простоя соединения.

You can also register AS400JDBCConnectionPoolDataSource objects by using a Java Naming and Directory Interface (JNDI) service provider. Более подробную информацию о провайдере JNDI можно найти в разделе IBM Toolbox for Java - Ссылки на справочную информацию.

Пример: Применение пула соединений

Ниже приведен пример получения источника данных пула соединений от JNDI и создания с его помощью пула из 10 соединений:

```
          // Получение объекта AS400JDBCConnectionPoolDataSource от JNDI
          // (предполагается, что среда JNDI уже настроена).
     Context context = new InitialContext(environment);
     AS400JDBCConnectionPoolDataSource datasource =
          (AS400JDBCConnectionPoolDataSource)context.lookup("jdbc/myDatabase");

          // Создание объекта AS400JDBCConnectionPool.
     AS400JDBCConnectionPool pool = new AS400JDBCConnectionPool(datasource);

          // Добавление в пул 10 соединений, которые могут применяться
          // приложениями (при этом создаются физические соединения с
          // базой данных с учетом заданного источника данных).
     pool.fill(10);

          // Получение ссылки на соединение с базой данных из пула.
     Connection connection = pool.getConnection();

     ... Выполнение различных операций с базой данных.

          // Закрытие ссылки на соединение для его возвращения в пул.
     connection.close();

     ... Применение других соединений пула.

     // Закрытие пула для освобождения всех ресурсов.
     pool.close();
```

AS400JDBCConnectionPool Javadoc

AS400JDBCConnectionPool Javadoc

AS400JDBCConnectionPoolDataSource Javadoc

AS400JDBCManagedConnectionPoolDataSource class:

The JDBC connection pool is a managed connection pool that allows you to reuse connections.

The AS400JDBCManagedConnectionPoolDataSource class simplifies connection pool maintenance by eliminating the need for user applications to implement their own management code. The connection pool is used in combination with the JDBC DataSource interface mechanism. The class AS400JDBCManagedConnectionPoolDataSource contains an implementation of the javax.sql.DataSource interface. This class manages the functionality of the connection pooling mechanism and allows you to access it like a normal DataSource. Setting up and using self-managed JDBC connection pool manager involves the usage of Java Naming and Directory Interface (JNDI) to bind the connection pool managed datasource. In order for self-managed JDBC connection pooling to function, JNDI is necessary.

These properties allow you to configure the attributes of the connection pool:

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

+ For more information, see the Javadocs on setX() methods where X is the property.

+ *Examples: Using AS400JDBCManagedConnectionPoolDataSource class:*

+ These examples demonstrate the use of the AS400JDBCManagedConnectionPoolDataSource class. The
+ AS400JDBCManagedConnectionPoolDataSource class simplifies connection pool maintenance by eliminating the
+ need for user applications to implement their own management code.

+ **Примечание:** By using the code examples, you agree to the terms of the “Лицензия на исходный код и отказ от
+ обязательств” на стр. 768.

+ **Пример 1**

+ This brief example shows the basic use of the AS400JDBCManagedConnectionPoolDataSource class.

```
+ import javax.naming.Context;
+ import javax.naming.InitialContext;
+ import javax.sql.DataSource;
+
+ import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
+ import com.ibm.as400.access.AS400JDBCManagedDataSource;
+
+
+ public class TestJDBCConnPoolSnippet
+ {
+
+     /**
+      * @param args
+      */
+     void test()
+     {
+         AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();
+
+         // Set general datasource properties.  Note that both connection pool datasource (CPDS) and managed
+                  // datasource (MDS) have these properties, and they might have different values.
+         cpds0.setServerName(host);
+         cpds0.setDatabaseName(host);//iasp can be here
+         cpds0.setUser(userid);
+         cpds0.setPassword(password);
+
```

```
+
+         cpds0.setSavePasswordWhenSerialized(true);
+
+         // Set connection pooling-specific properties.
+         cpds0.setInitialPoolSize(initialPoolSize_);
+         cpds0.setMinPoolSize(minPoolSize_);
+         cpds0.setMaxPoolSize(maxPoolSize_);
+         cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
+         cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
+         cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
+         //cpds0.setReuseConnections(false);  // do not re-use connections
+
+         // Set the initial context factory to use.
+
+         System.setProperty(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
+
+
+         // Get the JNDI Initial Context.
+         Context ctx = new InitialContext();
+
+         // Note: The following is an alternative way to set context properties locally:
+         //   Properties env = new Properties();
+         //   env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
+         //   Context ctx = new InitialContext(env);
+
+         ctx.rebind("mydatasource", cpds0);  // We can now do lookups on cpds, by the name "mydatasource".
+
+         // Create a standard DataSource object that references it.
+
+         AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
+         mds0.setDescription("DataSource supporting connection pooling");
+         mds0.setDataSourceName("mydatasource");
+         ctx.rebind("ConnectionPoolingDataSource", mds0);
+
+         DataSource dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
+
+         AS400JDBCManagedDataSource mds_ = (AS400JDBCManagedDataSource)dataSource_;
+
+         boolean isHealthy = mds_.checkPoolHealth(false);  //check pool health
+
+         Connection c = dataSource_.getConnection();
+
+     }
+
+ }
```

+ **Пример 2**

+ This example shows more details about how to use the AS400JDBCManagedConnectionPoolDataSource class.

```
+ import java.awt.TextArea;
+ import java.io.BufferedReader;
+ import java.io.File;
+ import java.io.FileReader;
+ import java.io.FileInputStream;
+ import java.io.FileOutputStream;
+ import java.io.OutputStream;
+ import java.io.PrintStream;
+ import java.util.Vector;
+ import java.util.Properties;
+
+ import java.sql.Connection;
+ import javax.sql.DataSource;
+ import java.sql.ResultSet;
+ import java.sql.Statement;
+ import javax.naming.*;
+ import java.util.Date;
+ import java.util.ArrayList;
+ import java.util.Random;
+ import com.ibm.as400.access.AS400;
+ import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
+ import com.ibm.as400.access.AS400JDBCManagedDataSource;
+ import com.ibm.as400.access.Trace;
+
+
+ public class TestJDBCConnPool
+ {
+
```

```
+     private static final boolean DEBUG = false;
+
+     // If you turn this flag on, be sure to also turn on the following flag:
+     //   AS400JDBCConnection.TESTING_THREAD_SAFETY.
+     private static final boolean TESTING_THREAD_SAFETY = false;
+
+     private static String userid;
+     private static String password;
+     private static String host;
+
+     // Note: For consistency, all time values are stored units of milliseconds.
+     private int initialPoolSize_;  // initial # of connections in pool
+     private int minPoolSize_;      // max # of connections in pool
+     private int maxPoolSize_;      // max # of connections in pool
+     private long maxLifetime_;     // max lifetime (msecs) of connections in pool
+     private long maxIdleTime_;     // max idle time (msecs) of available connections in pool
+     private long propertyCycle_;   // pool maintenance frequency (msecs)
+
+     private int numDaemons_;       // # of requester daemons to create
+     private static long timeToRunDaemons_; // total duration (msecs) to let the daemons run
+     private long daemonMaxSleepTime_;  // max time (msecs) for requester daemons to sleep each cycle
+     private long daemonMinSleepTime_;  // min time (msecs) for requester daemons to sleep each cycle
+     private long poolHealthCheckCycle_;  // # of msecs between calls to checkPoolHealth()
+
+     private boolean keepDaemonsAlive_ = true;  // When this is false, the daemons shut down.
+
+     private DataSource dataSource_;
+     private AS400JDBCManagedDataSource mds_;
+
+     private final Object daemonSleepLock_ = new Object();
+
+     private Random random_ = new Random();
+
+     static
+     {
+         try {
+             Class.forName("com.ibm.as400.access.AS400JDBCDriver");
+         }
+         catch (Exception e) {
+             System.out.println("Unable to register JDBC driver.");
+             System.exit(0);
+         }
+     }
+
+
+
+     public static void main(String[] args)
+     {
+         host = args[0];
+         userid = args[1];
+         password = args[2];
+         timeToRunDaemons_ = (new Integer(args[3])).intValue() * 1000; //milliseconds
+         //args[3]=time to run in seconds
+         TestJDBCConnPool cptest = new TestJDBCConnPool();
+
+         cptest.setup();
+         cptest.runTest();
+     }
+
+
+
+
+     public void setup()
+     {
+         try
+         {
+             if (DEBUG) System.out.println("TESTING_THREAD_SAFETY flag is " + (TESTING_THREAD_SAFETY
+ ? "true" : "false"));
+
+             if (TESTING_THREAD_SAFETY)
+             {
+                 // Adjust values for performing thread-intensive stress testing.
+                 // NOTE: This assumes that the AS400JDBCConnection class has also been modified to
+                                       // not make actual connections to an actual server.
+                 // To do this, edit AS400JDBCConnection.java, changing its TESTING_THREAD_SAFETY
+                                       // flag to 'false', and recompile.
+                 minPoolSize_ = 100;
+                 maxPoolSize_ = 190;
+                 initialPoolSize_ = 150;  // this should get reset to maxPoolSize_
```

```
+                 numDaemons_ = 75;
+                 if (timeToRunDaemons_ == 0) {
+                     timeToRunDaemons_ = 180*1000; // 180 seconds == 3 minutes
+                 }
+             }
+             else
+             {  // Set more conservative values, as we'll be making actual connections to an
+                // actual server, and we don't want to monopolize the server.
+                 minPoolSize_ = 5;
+                 maxPoolSize_ = 15;
+                 initialPoolSize_ = 9;
+                 numDaemons_ = 4;
+                 if (timeToRunDaemons_ == 0) {
+                     timeToRunDaemons_ = 15*1000; // 15 seconds
+                 }
+             }
+             maxLifetime_ = (int)timeToRunDaemons_ / 3;
+             maxIdleTime_ = (int)timeToRunDaemons_ / 4;
+             propertyCycle_ = timeToRunDaemons_ / 4;
+             poolHealthCheckCycle_ = Math.min(timeToRunDaemons_ / 4, 20*60*1000);
+             // at least once every 20 minutes (more frequently if shorter run-time)
+             daemonMaxSleepTime_ = Math.min(timeToRunDaemons_ / 3, 10*1000);
+             // at most 10 seconds (less if shorter run-time)
+             daemonMinSleepTime_ = 20;  // milliseconds
+
+             if (DEBUG) System.out.println("setup: Constructing
+ AS400JDBCManagedConnectionPoolDataSource (cpds0)");
+             AS400JDBCManagedConnectionPoolDataSource cpds0 = new
+ AS400JDBCManagedConnectionPoolDataSource();
+
+             // Set general datasource properties.  Note that both CPDS and MDS have these
+             // properties, and they might have different values.
+             cpds0.setServerName(host);
+             cpds0.setDatabaseName(host);//iasp can be here
+             cpds0.setUser(userid);
+             cpds0.setPassword(password);
+
+
+             cpds0.setSavePasswordWhenSerialized(true);
+
+             // Set connection pooling-specific properties.
+             cpds0.setInitialPoolSize(initialPoolSize_);
+             cpds0.setMinPoolSize(minPoolSize_);
+             cpds0.setMaxPoolSize(maxPoolSize_);
+             cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
+             cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
+             cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
+             //cpds0.setReuseConnections(false);  // don't re-use connections
+
+             // Set the initial context factory to use.
+
+
+             System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
+ "com.sun.jndi.fscontext.RefFSContextFactory");
+
+
+             // Get the JNDI Initial Context.
+             Context ctx = new InitialContext();
+
+             // Note: The following is an alternative way to set context properties locally:
+             //   Properties env = new Properties();
+             //   env.put(Context.INITIAL_CONTEXT_FACTORY,
+ "com.sun.jndi.fscontext.RefFSContextFactory");
+             //   Context ctx = new InitialContext(env);
+
+
+             ctx.rebind("mydatasource", cpds0);
+                         // We can now do lookups on cpds, by the name"mydatasource".
+
+             if (DEBUG) System.out.println("setup: lookup(\"mydatasource\"" + ")");
+          //   AS400JDBCManagedConnectionPoolDataSource cpds1 =
+ (AS400JDBCManagedConnectionPoolDataSource)ctx.lookup("mydatasource");
+            // if (DEBUG) System.out.println("setup: cpds1.getUser() == |" + cpds1.getUser() + "|");
+
+
+             // Create a standard DataSource object that references it.
+
+             if (DEBUG) System.out.println("setup: Constructing AS400JDBCManagedDataSource (mds0)");
+             AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
```

```
+             mds0.setDescription("DataSource supporting connection pooling");
+             mds0.setDataSourceName("mydatasource");
+             ctx.rebind("ConnectionPoolingDataSource", mds0);
+
+             if (DEBUG) System.out.println("setup: lookup(\"ConnectionPoolingDataSource\"" + ")");
+             dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
+             //dataSource_.setLogWriter(output_);
+             if (DEBUG) System.out.println("setup: dataSource_.getUser() == |" +
+ ((AS400JDBCManagedDataSource)dataSource_).getUser() + "|");
+
+             mds_ = (AS400JDBCManagedDataSource)dataSource_;
+
+
+         }
+         catch (Exception e)
+         {
+             e.printStackTrace();
+             System.out.println("Setup error during Trace file creation.");
+         }
+     }
+
+     void displayConnectionType(Connection conn, boolean specifiedDefaultId)
+     {
+         if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
+         {
+             System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "P)");
+         }
+         else
+         {
+             System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "NP)");
+         }
+     }
+
+
+     /**
+      *  Gets and returns connections from and to a connection pool for a while.
+      **/
+     public void runTest()
+     {
+         boolean ok = true;
+         try
+         {
+             System.out.println("Started test run at " + new Date());
+
+             if (DEBUG) System.out.println("Checking health just after datasource creation (we expect
+ that the pool does not exist yet) ...");
+             if (mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool exists prior to first getConnection().");
+             }
+
+             // Verify some setters/getters for JDBC properties.
+             System.out.println("Verifying setters/getters ...");
+
+             mds_.setAccess("read only");
+             if (!mds_.getAccess().equals("read only")) {
+                 ok = false;
+                 System.out.println("\nERROR: getAccess() returned unexpected value:
+ |"+mds_.getAccess()+"|");
+             }
+
+             boolean oldBool = mds_.isBigDecimal();
+             boolean newBool = (oldBool ? false : true);
+             mds_.setBigDecimal(newBool);
+             if (mds_.isBigDecimal() != newBool) {
+                 ok = false;
+                 System.out.println("\nERROR: isBigDecimal() returned unexpected value:
+ |"+mds_.isBigDecimal()+"|");
+             }
+             mds_.setBigDecimal(oldBool);
+
+             int oldInt = mds_.getBlockCriteria();
+             int newInt = (oldInt == 2 ? 1 : 2);
+             mds_.setBlockCriteria(newInt);
+             if (mds_.getBlockCriteria() != newInt) {
+                 ok = false;
+                 System.out.println("\nERROR: getBlockCriteria() returned unexpected value:
+ |"+mds_.getBlockCriteria()+"|");
+             }
```

```
+             mds_.setBlockCriteria(oldInt);
+
+             // Verify some setters and getters for socket properties.
+
+             oldBool = mds_.isKeepAlive();
+             newBool = (oldBool ? false : true);
+             mds_.setKeepAlive(newBool);
+             if (mds_.isKeepAlive() != newBool) {
+                 ok = false;
+                 System.out.println("\nERROR: isKeepAlive() returned unexpected value:
+ |"+mds_.isKeepAlive()+"|");
+             }
+             mds_.setKeepAlive(oldBool);
+
+             oldInt = mds_.getReceiveBufferSize();
+             newInt = (oldInt == 256 ? 512 : 256);
+             mds_.setReceiveBufferSize(newInt);
+             if (mds_.getReceiveBufferSize() != newInt) {
+                 ok = false;
+                 System.out.println("\nERROR: getReceiveBufferSize() returned unexpected value:
+ |"+mds_.getReceiveBufferSize()+"|");
+             }
+             mds_.setReceiveBufferSize(oldInt);
+
+
+             System.out.println("CONNECTION 1");
+             Object o = dataSource_.getConnection();
+            System.out.println(o.getClass());
+            System.out.println("******LOOK ABOVE*******");
+             Connection c1 = dataSource_.getConnection();
+
+             if (DEBUG) displayConnectionType(c1, true);
+
+             if (DEBUG) System.out.println("Checking health after first getConnection() ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after first getConnection().");
+             }
+
+             if (!TESTING_THREAD_SAFETY)
+             {
+                 try
+                 {
+                     c1.setAutoCommit(false);
+                     if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
+                     Statement s = c1.createStatement();
+                     ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
+                     while (rs.next ()) {
+                         if (DEBUG) System.out.println(rs.getString(2));
+                     }
+                     rs.close();
+                                  s.close();
+                 }
+                 catch (Exception e) {
+                     e.printStackTrace();
+                     if (DEBUG) System.out.println("Checking health after fatal connection error
+ ...");
+                     if (!mds_.checkPoolHealth(true)) {
+                         ok = false;
+                         System.out.println("\nERROR: Pool is not healthy after fatal connection
+ error.");
+                     }
+                 }
+             }
+
+
+             System.out.println("CONNECTION 2");
+             Connection c2 = dataSource_.getConnection(userid, password);
+             if (DEBUG) displayConnectionType(c2, false);
+             System.out.println("CONNECTION 3");
+             Connection c3 = dataSource_.getConnection();
+             if (DEBUG) displayConnectionType(c3, true);
+             c1.close();
+
+             if (DEBUG) System.out.println("Checking health after first close() ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after first close().");
+             }
```

```
+
+             System.out.println("CONNECTION 4");
+             Connection c4 = dataSource_.getConnection();
+             if (DEBUG) displayConnectionType(c4, true);
+
+             c1.close();  // close this one again
+             c2.close();
+             c3.close();
+             c4.close();
+
+             if (DEBUG) System.out.println("Checking health after last close() ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after last close().");
+             }
+
+             // Start the test daemons.
+             System.out.println("Starting test daemons");
+             startThreads();
+
+             // Run the test daemons for a while; check pool health periodically.
+
+             long startTime = System.currentTimeMillis();
+             long endTime = startTime + timeToRunDaemons_;
+             while (System.currentTimeMillis() < endTime)
+             {
+                 System.out.print("h");
+                 // Let the daemons run for a while, then check pool health.
+                 try {
+                     Thread.sleep(poolHealthCheckCycle_);
+                 }
+                 catch (InterruptedException ie) {}
+                 if (!mds_.checkPoolHealth(true)) {
+                     ok = false;
+                     System.out.println("\nERROR: Pool is not healthy after test daemons started.");
+                 }
+             }
+
+             // Stop the test daemons.
+             System.out.println("\nStopping test daemons");
+             stopThreads();
+
+             if (DEBUG) System.out.println("Checking health after connectionGetter daemons have run
+ ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after test daemons stopped.");
+             }
+
+
+             if (!TESTING_THREAD_SAFETY)
+             {
+                 System.out.println("CONNECTION 5");
+                 Connection c = dataSource_.getConnection();
+                 if (DEBUG) displayConnectionType(c, true);
+                 c.setAutoCommit(false);
+                 if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
+                 Statement s = c.createStatement();
+                 ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
+                 while (rs.next ()) {
+                     if (DEBUG) System.out.println(rs.getString(2));
+                 }
+                 rs.close();
+                                 s.close();
+                 c.close();            }
+
+
+
+             System.out.println("\nClosing the pool...");
+             mds_.closePool();
+
+             if (DEBUG) System.out.println("Checking health after pool closed ...");
+             Trace.setTraceJDBCOn(true);  // make sure the final stats get printed out
+             Trace.setTraceOn(true);
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after pool closed.");
+             }
+
```

```
+             System.out.println();
+             if(ok==true)
+                 System.out.println("test ran ok");
+             else
+                 System.out.println("test failed");
+
+         }
+         catch (Exception e)
+         {
+             System.out.println(e);
+             e.printStackTrace();
+         }
+         finally
+         {
+             System.out.println("Ended test at " + new Date());
+         }
+     }
+
+
+     void startThreads()
+     {
+
+         // Create a bunch of threads that call getConnection().
+         Thread[] threads = new Thread[numDaemons_];
+         for (int i=0; i<numDaemons_; i++)
+         {
+             ConnectionGetter getter;
+             // Flip a coin to see if this daemon will specify the default uid, or unique uid.
+             if (random_.nextBoolean())
+             {
+                 getter = new ConnectionGetter(userid,password);
+                 if (TESTING_THREAD_SAFETY) {  // we can use fictional userid
+                     getter = new ConnectionGetter("Thread"+i, "Pwd"+i);
+                 }
+                 else {  // must use a real userid
+                     getter = new ConnectionGetter(userid,password);
+                 }
+             }
+             else
+             {
+                 getter = new ConnectionGetter(null, null);
+             }
+             threads[i] = new Thread(getter, "["+i+"]");
+             threads[i].setDaemon(true);
+         }
+
+         // Start the threads.
+         for (int i=0; i<numDaemons_; i++)
+         {
+             threads[i].start();
+         }
+     }
+
+     void stopThreads()
+     {
+         // Tell the threads to stop.
+         keepDaemonsAlive_ = false;
+         synchronized (daemonSleepLock_) {
+             daemonSleepLock_.notifyAll();
+         }
+
+         // Wait for the daemons to stop.
+         try {
+             Thread.sleep(3000);
+         }
+         catch (InterruptedException ie) {}
+     }
+
+
+
+     // ConnectionGetter ----------------------------------------------------------------
+
+     /**
+          Helper class. This daemon wakes up at random intervals and either
+          gets another connection from the connection pool or returns a previously-gotten connection
+  to the pool.
+       **/
+     private final class ConnectionGetter implements Runnable
```

```
+     {
+         private String uid_;
+         private String pwd_;
+         private boolean useDefaultUid_;
+         private long maxSleepTime_;
+         private String threadName_;
+         private boolean firstConnection_ = true;
+         ArrayList connections_ = new ArrayList();
+         // list of connections that this getter currently 'owns'.
+
+         ConnectionGetter(String uid, String pwd) {
+             uid_ = uid;
+             pwd_ = pwd;
+             if (uid_ == null) useDefaultUid_ = true;
+             else useDefaultUid_ = false;
+             maxSleepTime_ = daemonMaxSleepTime_;  // our own copy that we can adjust
+         }
+
+         public void run( )
+         {
+             threadName_ = Thread.currentThread().getName();
+             if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Starting up");
+
+             try
+             {
+                 while (keepDaemonsAlive_)
+                 {
+                     try
+                     {
+                         // Pick a random sleep-time, between min and max values.
+                         long sleepTime = Math.max((long)(maxSleepTime_ * random_.nextFloat()),
+                                 daemonMinSleepTime_);
+                         // Note: Must never call wait(0), because that waits forever.
+                         synchronized (daemonSleepLock_) {
+                             try {
+                                 daemonSleepLock_.wait(sleepTime);
+                                 System.out.print(".");
+                             }
+                             catch (InterruptedException ie) {}
+                         }
+                         if (!keepDaemonsAlive_) break;
+
+                         // Decide by chance whether to request another connection or return a
+                         // previously-obtained connection.
+                         Connection conn;
+                         if (random_.nextBoolean())  // Leave the decision to chance.
+                         { // Request another connection.
+                             if (useDefaultUid_)
+                             {
+                                 if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") -
+ get()");
+                                 conn = dataSource_.getConnection();
+                             }
+                             else
+                             {
+                                 if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") -
+ get("+uid_+",***)");
+                                 conn = dataSource_.getConnection(uid_, pwd_);
+                             }
+
+                             if (conn == null) {
+                                 System.out.println("ConnectionGetter("+threadName_+") ERROR:
+ getConnection() returned null");
+                             }
+                             else
+                             {
+                                 // Occasionally "forget" that we own a connection, and neglect to
+                                 // close it.
+                                 // Orphaned connections should eventually exceed their maximum
+                                 // lifetime and get "reaped" by the connection manager.
+                                 float val = random_.nextFloat();
+                                 if (firstConnection_ || val < 0.1) { // 'orphan' a few gotten
+ соединения                                          firstConnection_ = false;
+                                 }
+                                 else {
+                                     connections_.add(conn);
+                                 }
+                                 if (DEBUG) displayConnectionType(conn, useDefaultUid_);
+                                 if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
```

```
+                                {  // We got a pooled connection.  Try speeding up our cycle time.
+                                    if (maxSleepTime_ &gt; 100) maxSleepTime_--;
+                                    else maxSleepTime_ = 100;
+                                }
+                                else
+                                {  // We didn't get a pooled connection.  That means that the pool
+                                   // must be at capacity.  Slow down our cycle time a bit.
+                                    maxSleepTime_ = maxSleepTime_ + 50;
+                                }
+                            }
+                        }
+                        else { // Close a connection that we currently own.
+                            if (connections_.size() != 0) {
+                                conn = (Connection)connections_.remove(0);
+                                conn.close();
+                            }
+                        }
+
+                    }  // inner try
+                    catch (Exception e)
+                    {
+                        e.printStackTrace();
+                    }
+                } // outer while
+            }  // outer try
+            finally
+            {
+                if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Stopping");
+                // Return all the connections that I still have in my list.
+                for (int i=0; i<connections_.size(); i++) {
+                    Connection conn = (Connection)connections_.remove(0);
+                    try { conn.close(); } catch (Exception e) { e.printStackTrace(); }
+                }
+
+            }
+        }
+
+    }  // internal class 'ConnectionGetter'
+
+
+ }
```

+ **Интерфейс DatabaseMetaData:**

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

Ниже приведен пример получения списка таблиц, который называется каталогом:

```
                    // Подключение к серверу.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Получение метаданных базы данных с помощью соединения.
DatabaseMetaData dbMeta = c.getMetaData();

                    // Получение списка таблиц, соответствующих следующим критериям.
String catalog = "myCatalog";
String schema  = "mySchema";
String table   = "myTable%"; // % задает шаблон для поиска
String types[]  = {"TABLE", "VIEW", "SYSTEM TABLE"};
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

                    // Получение значений с помощью итераций по ResultSet.

                    // Закрытие объекта Connection.
c.close();
```

Информация, связанная с данной

AS400JDBCDatabaseMetaData Javadoc

Класс AS400JDBCDataSource:

The AS400JDBCDataSource class represents a factory for System i5 database connections. The AS400JDBCConnectionPoolDataSource class represents a factory for AS400JDBCPooledConnection objects.

Для регистрации объектов источников данных обоих типов служит провайдер Java Naming and Directory Interface (JNDI). For more information about JNDI service providers, see “Related information for IBM Toolbox for Java” на стр. 765.

Примеры

Ниже приведены примеры создания и использования объектов AS400JDBCDataSource. В последних двух примерах проиллюстрирована процедура регистрации объекта AS400JDBCDataSource с помощью JNDI и применения объекта, полученного от JNDI, для создания соединения с базой данных. Обратите внимание, что исходный код примеров практически не зависит от того, с каким поставщиком услуг JNDI вы работаете.

Пример: Создание объекта AS400JDBCDataSource

В приведенном ниже примере описана процедура создания объекта AS400JDBCDataSource и подключения к базе данных:

```
// Создание источника данных для установления соединения.
AS400JDBCDataSource datasource = new AS400JDBCDataSource("myAS400");
datasource.setUser("myUser");
datasource.setPassword("MYPWD");

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

Пример: Создание объекта AS400JDBCConnectionPoolDataSource, предназначенного для кэширования соединений JDBC

В приведенном ниже примере показано, каким образом объект AS400JDBCConnectionPoolDataSource может применяться для кэширования соединений JDBC.

```
// Создание источника данных для установления соединения.
AS400JDBCConnectionPoolDataSource dataSource = new AS400JDBCConnectionPoolDataSource("myAS400");
datasource.setUser("myUser");
datasource.setPassword("MYPWD");

// Получение соединения из пула.
PooledConnection pooledConnection = datasource.getPooledConnection();
```

Пример: Хранение объекта AS400JDBCDataSource с помощью классов поставщика услуг JNDI

Ниже приведен пример того, как с помощью классов поставщика услуг JNDI можно хранить объект DataSource непосредственно в интегрированной файловой системе сервера:

```
R       // Create a data source to the System i5 database.
R       AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
R       dataSource.setServerName("myAS400");
R       dataSource.setDatabaseName("myAS400 Database");
R
R       // Регистрация источника данных с помощью JNDI.
R       Hashtable env = new Hashtable();
R       env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
R       Context context = new InitialContext(env);
R       context.bind("jdbc/customer", dataSource);
R
R       // Получение от JNDI объекта AS400JDBCDataSource и создание соединения.
R       AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup("jdbc/customer");
R       Connection connection = datasource.getConnection("myUser", "MYPWD");
```

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

В приведенном ниже примере описана процедура сохранения объекта с помощью классов IBM SecureWay Directory на сервере LDAP:

```
R       // Create a data source to the System i5 database.
R       AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
R       dataSource.setServerName("myAS400");
R       dataSource.setDatabaseName("myAS400 Database");
R
R       // Регистрация источника данных с помощью JNDI.
R       Hashtable env = new Hashtable();
R       env.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.jndi.LDAPCtxFactory");
R       Context context = new InitialContext(env);
R       context.bind("cn=myDatasource, cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion",
R                    dataSource);
R // Получение от JNDI объекта AS400JDBCDataSource и создание соединения.
R       AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup(
R          "cn=myDatasource, cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion");
R       Connection connection = datasource.getConnection("myUser", "MYPWD");
```

AS400JDBCDataSource Javadoc

AS400JDBCConnectionPoolDataSource Javadoc

AS400JDBCPooledConnection Javadoc

Регистрация драйвера JDBC:

Before using JDBC to access data in a server database file, you need to register the JDBC driver for the IBM Toolbox for Java licensed program with the DriverManager.

Это можно сделать в программе на Java или с помощью системного свойства Java.

- Регистрация с помощью системного свойства

  В каждой виртуальной машине применяется собственный метод настройки системных свойств. Например, в JDK нужно вызвать команду Java с опцией -D. Для того чтобы задать драйвер в системных свойствах, введите:

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

- Регистрация с помощью программы на Java

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

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

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

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

The IBM Toolbox for Java JDBC driver does not require an AS400 object as an input parameter like the other IBM Toolbox for Java classes that get data from a server. Тем не менее, объект AS400 применяется для настройки ИД пользователя по умолчанию и кэширования пароля. При первом подключении к серверу обычно запрашивается ИД пользователя и пароль. Пользователь может сохранить ИД в качестве ИД по умолчанию и занести пароль в кэш паролей. As in the other IBM Toolbox for Java functions, if the user ID and password are supplied by the Java program, the default user is not set and the password is not cached. See “Managing connections in Java programs” на стр. 450 for information about managing connections.

Подключение к базе данных сервера с помощью драйвера JDBC

Для подключения к базе данных сервера можно воспользоваться методом DriverManager.getConnection(). В качестве параметра методу DriverManager.getConnection() передается строка URL. После этого администратор драйвера JDBC пытается найти драйвер, который может подключиться к базе данных с заданным URL. When using the IBM Toolbox for Java driver, use the following syntax for the URL:

```
"jdbc:as400://имя-системы/схема-по-умолчанию;список-свойств"
```

Примечание: Из ULR можно исключить значения systemName или defaultSchema.

Если вы планируете применять паспорт Kerberos, укажите в объекте URL JDBC только имя системы (без пароля). Имя пользователя будет получено с помощью Общих служб защиты Java (JGSS), поэтому его не нужно указывать в URL JDBC. В объекте AS400JDBCConnection можно задать только один способ идентификации. Если вы зададите пароль, то все паспорта и разрешения Kerberos будут удалены. For more information, see “Класс AS400” на стр. 21 and J2SDK, v1.4 Security Documentation .

Пример: Подключение к серверу с помощью драйвера JDBC

Пример: Использование URL, в котором не задано имя системы

В этом примере пользователю будет предложено указать имя системы, к которой ему необходимо подключиться.

```
                    // Подключение к безымянной системе.
                    // Запрос имени системы у пользователя.
Connection c = DriverManager.getConnection("jdbc:as400:");
```

Пример: Подключение к базе данных сервера; схема и свойства по умолчанию не заданы

```
                    // Connect to system 'mySystem'. Нет                        // Не задана схема по умолчанию и
                    // и свойства системы.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");
```

Пример: Подключение к базе данных сервера; схема и свойства по умолчанию заданы

```
                    // Connect to system 'mySys2'.                          // Задана схема по умолчанию
                    // и свойства системы.
Connection c2 = DriverManager.getConnection("jdbc:as400://mySys2/mySchema");
```

Пример: Подключение к базе данных сервера и указание свойств с помощью интерфейса java.util.Properties

Программа на Java может передать набор свойств JDBC с помощью интерфейса java.util.Properties или задать их в URL. Список поддерживаемых свойств приведен в разделе “IBM Toolbox for Java JDBC properties” на стр. 328.

Например, ниже приведен фрагмент программы, в котором свойства задаются с помощью интерфейса Properties.

```
                    // Создание объекта свойств.
 Properties p = new Properties();

                    // Настройка свойств для
                    // соединения.
 p.put("naming", "sql");
 p.put("errors", "full");

                    // Подключение с помощью объекта
                    //
 Connection c = DriverManager.getConnection("jdbc:as400://mySystem",p);
```

Пример: Подключение к базе данных сервера и указание свойств с помощью адреса URL

```
                    // Connect using properties.                        // Свойства задаются не в объекте свойств,
                    // а в URL.
                    //
 Connection c = DriverManager.getConnection(
                     "jdbc:as400://mySystem;naming=sql;errors=full");
```

Пример: Подключение к базе данных сервера и указание ИД пользователя и пароля

```
                    // Подключение к системе; свойства заданы
                    // в URL; указываются ИД пользователя и
                    // пароль.
```

```
Connection c = DriverManager.getConnection(
                    "jdbc:as400://mySystem;naming=sql;errors=full",
                    "auser",
                    "apassword");
```

Пример: Отключение от базы данных

Для отключения от сервера применяется метод close() объекта Connecting. Закрыть соединение, созданное в предыдущем примере, можно с помощью следующего оператора:

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

AS400JDBCDriver Javadoc

Класс AS400JDBCParameterMetaData:

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

Методы класса AS400JDBCParameterMetaData позволяют выполнять следующие операции:

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

Пример: Применение класса AS400JDBCParameterMetaData

В следующем примере показан один из способов применения класса AS400JDBCParameterMetaData для получения параметров динамически создаваемого объекта PreparedStatement:

```
// Подключение через драйвер.
Class.forName("com.ibm.as400.access.AS400JDBCDriver");
Connection connection =
   DriverManager.getConnection("jdbc:as400://myAS400", "myUserId", "myPassword");

// Создание подготовленного оператора.
PreparedStatement ps =
   connection.prepareStatement("SELECT STUDENTS FROM STUDENTTABLE WHERE STUDENT_ID= ?");

// ИД студента заносится в параметр 1.
ps.setInt(1, 123456);

// Получение метаданных параметров подготовленного оператора.
ParameterMetaData pMetaData = ps.getParameterMetaData();

// Получение числа параметров подготовленного оператора.
// Метод возвращает значение 1.
int parameterCount = pMetaData.getParameterCount();

// Определение типа параметра 1.
// Метод возвращает значение INTEGER.
String getParameterTypeName = pMetaData.getParameterTypeName(1);
```

Информация, связанная с данной

AS400JDBCParameterMetaData Javadoc

Интерфейс PreparedStatement:

You can use a PreparedStatement object when an SQL statement is going to be run many times. A prepared statement is an SQL statement that has been precompiled.

Такие операторы рекомендуется применять вместо объекта Statement, компилирующего оператор при каждом вызове. Кроме того, у оператора SQL в объекте PreparedStatement может быть несколько входных параметров. Для создания объектов PreparedStatement предназначен метод Connection.prepareStatement().

Объект PreparedStatement позволяет объединить несколько команд SQL в одну группу и передать их на обработку в базу данных в пакетном режиме. Выполнение группы операторов в пакетном режиме обычно занимает меньше времени, чем выполнение операторов по отдельности, что позволяет повысить производительность. Дополнительная информация о поддержке пакетного режима приведена в разделе Изменения, внесенные в поддержку JDBC.

Пример: Применение интерфейса PreparedStatement

Ниже приведен пример работы с интерфейсом PreparedStatement.

```
                    // Подключение к серверу.
    Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Создание объекта PreparedStatement.
                    // Он выполняет предварительную обработку
                    // specified SQL statement.                        // отмечена позиция, в которой должен быть
                    // задан параметр перед запуском
                    // оператора.
    PreparedStatement ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");
// Запуск оператора с
                    // заданными параметрами.
    ps.setString(1, "JOSH");
    ps.setInt(2, 789);
    ps.executeUpdate();

                    // Запуск оператора с
                    // заданными параметрами.
    ps.setString(1, "DAVE");
    ps.setInt(2, 456);
    ps.executeUpdate();

                    // Закрытие объектов PreparedStatement и
                    // Connection.
    ps.close();
    c.close();
```

Информация, связанная с данной

AS400JDBCPreparedStatement Javadoc

Класс ResultSet:

You can use a ResultSet object to access a table of data that was generated by running a query. Строки таблицы просматриваются последовательно. Поля одной строки можно просматривать в любом порядке.

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

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

Перемещение курсора

Курсор - это внутренний указатель на строку таблицы результатов, к которой в данный момент обратилась программа на Java.

Производительность метода getRow() возросла. В версиях младше V5R2 после вызова метода ResultSet.last(), ResultSet.afterLast() или ResultSet.absolute() с отрицательным параметром номер текущей строки становился недоступным. В текущей версии это ограничение снято, что позволяет использовать все возможности метода getRow().

JDBC 2.0 и более поздние спецификации JDBC предоставляют дополнительные методы для перемещения по базе данных:

Перемещение курсора путем прокрутки	
absolute afterLast beforeFirst first getRow isAfterLast isBeforeFirst	isFirst isLast last moveToCurrentRow moveToInsertRow previous relative

Функция прокрутки

Записи таблицы результатов, созданной в результате выполнения оператора, можно просматривать (прокручивать) от начала к концу или от конца к началу.

Таблица результатов, позволяющая перемещаться по записям таким образом, называется таблицей с возможностью прокрутки. Такие таблицы поддерживают абсолютную позицию курсора. Кроме того, вы можете перейти к строке, указав ее номер (абсолютную позицию курсора).

JDBC 2.0 и более поздние спецификации JDBC предоставляют две дополнительные функции прокрутки, которые можно применять при работе с классом ResultSet: прокрутка без учета изменений и прокрутка с учетом изменений.

В отличие от прокрутки с учетом изменений, прокрутка без учета изменений обычно не учитывает те изменения, которые были внесены в базу данных за время работы с таблицей результатов.

R **Примечание:** System i5 only allows read-only access for scrollable insensitive cursors. IBM Toolbox for Java
R supports a scroll-insensitive cursor if the result set concurrency is read-only. Если таблица
R результатов работает в режиме без учета изменений и режим можно изменить, таблица
R переводится в режим работы с учетом изменений и пользователю отправляется
R предупреждение.

Таблицы результатов с возможностью обновления

В приложениях могут применяться таблицы результатов, доступные только для чтения (в данные нельзя вносить изменения) или для изменения (разрешено изменение данных; для управления доступом других транзакций к базе данных может применяться блокировка на запись). В таблице результатов с возможностью обновления можно изменять, вставлять и удалять строки. Существует много различных методов обновления, в том числе:

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

See Method Summary in the AS400JDBCResultSet Javadoc for a complete listing of the update methods available through the ResultSet interface.

Пример: Таблицы результатов с возможностью обновления

Ниже приведен пример таблицы с возможностью обновления (update) и внесения изменений в открытую таблицу результатов (scroll sensitive).

```
                    // Подключение к серверу.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Создание объекта Statement. Набор результатов устанавливается
                    // доступным для обновления.
Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

                    // Запуск запроса. Результат запроса помещается
                    // в объект ResultSet.
ResultSet rs = s.executeQuery ("SELECT NAME,ID FROM MYLIBRARY.MYTABLE FOR UPDATE");

                    // Просмотр строк таблицы результатов.
                    // В каждой строке старый ИД заменяется
                    // на новый.
int newId = 0;
while (rs.next ())
{

                    // Получение значений из ResultSet.
                    // Первое значение - строка,
                    // а второе - целое число.
    String name = rs.getString("NAME");
    int id = rs.getInt("ID");

    System.out.println("Имя = " + name);
    System.out.println("Старый ИД = " + id);

                    // Обновление целочисленного ИД.
    rs.updateInt("ID", ++newId);

                    // Запись обновлений на сервер.
    rs.updateRow ();

    System.out.println("Новый ИД = " + newId);
}

                    // Закрытие объектов Statement и
                    // Connection.
s.close();
c.close();
```

Интерфейс ResultSetMetaData

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

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

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

Кроме того, при установке этого свойства включается поддержка метода ResultSetMetaData.getSchemaName(int). Обратите внимание, что применение расширенных метаданных может привести к снижению производительности, так как с сервера будет загружаться больший объем информации.

AS400JDBCResultSet Javadoc

AS400ResultSetMetaData Javadoc

Класс AS400JDBCRowSet:

The AS400JDBCRowSet class represents a connected rowset that encapsulates a JDBC result set. The methods on AS400JDBCRowSet are very similar to those of the AS400JDBCResultSet. При вызове этих методов устанавливается соединение с базой данных.

С помощью экземпляров класса AS400JDBCDataSource или AS400JDBCConnectionPoolDataSource можно создать соединение с базой данных, которая будет применяться при работе с классом AS400JDBCRowSet.

Примеры

Ниже приведены примеры применения класса AS400JDBCRowSet:

Пример: Создание, заполнение и обновление объекта AS400JDBCRowSet:

```
DriverManager.registerDriver(new AS400JDBCDriver());
// Установить соединение с помощью URL.
AS400JDBCRowSet rowset = new AS400JDBCRowSet("jdbc:as400://mySystem","myUser", "myPassword");

// Настройка команды для заполнения списка.
rowset.setCommand("SELECT * FROM MYLIB.DATABASE");

// Заполнение набора строк.
rowset.execute();

// Обновить балансы заказчика.
while (rowset.next())
{
   double newBalance = rowset.getDouble("BALANCE") +
                       july_statements.getPurchases(rowset.getString("CUSTNUM"));
rowset.updateDouble("BALANCE", newBalance);
   rowset.updateRow();
}
```

Пример: Создание и наполнение объекта AS400JDBCRowSet с помощью данных, полученных из источника в JNDI

```
// Получить зарегистрированный в JNDI источник данных (предполагается, что среда JNDI настроена).
Context context = new InitialContext();
AS400JDBCDataSource dataSource = (AS400JDBCDataSource) context.lookup("jdbc/customer");

AS400JDBCRowSet rowset = new AS400JDBCRowSet();
// Установить соединение, задав имя источника данных.
rowset.setDataSourceName("jdbc/customer");
rowset.setUsername("myuser");
rowset.setPassword("myPasswd");

// Настройка подготовленного оператора и инициализация параметров.
rowset.setCommand("SELECT * FROM MYLIBRARY.MYTABLE WHERE STATE = ? AND BALANCE > ?");
rowset.setString(1, "MINNESOTA");
rowset.setDouble(2, MAXIMUM_LIMIT);

// Заполнение набора строк.
rowset.execute();
```

AS400JDBCRowSet Javadoc

AS400JDBCResultSet Javadoc

Класс AS400JDBCSavepoint:

The IBM Toolbox for Java AS400JDBCSavepoint class represents a logical breaking point in a transaction. Применение точек сохранения позволяет минимизировать число изменений, которые требуется отменить при откате транзакции.

Рисунок 1: Применение точек сохранения для управления откатами транзакций

Например, на рисунке 1 показана транзакция, содержащая две точки сохранения, A и B. При откате транзакции к точке сохранения отменяются все изменения, внесенные с момента отката до точки сохранения. Все остальные изменения, внесенные при выполнении транзакции, остаются в силе. Обратите внимание, что после выполнения отката до точки сохранения A вы не сможете выполнить откат до точки сохранения B. Точка сохранения B будет недоступна после того, как будет выполнен откат к более ранней точке сохранения.

Пример: Работа с точками сохранения

В этом сценарии рассматривается приложение, обновляющее базу данных студентов. После обновления поля во всех записях о студентах была выполнена фиксация. Программа обнаружила ошибку, связанную с обновлением поля, и выполнила откат внесенных изменений. Вам известно, что эта ошибка связана только с обработкой текущей записи.

В результате после обновления каждой записи о студенте была установлена точка сохранения. Теперь при повторном возникновении ошибки потребуется откатить только последнюю операцию обновления таблицы студентов. Таким образом, будет выполнен откат не всей операции, а ее небольшого фрагмента.

Следующий пример программы иллюстрирует возможности применения точек сохранения. В программе предполагается, что ИД студента Джон равен 123456, а ИД студентки Джейн равен 987654.

```
// Подключение через драйвер
Class.forName("com.ibm.as400.access.AS400JDBCDriver");

// Создание объекта statement
Statement statement = connection.createStatement();

// Добавление в запись Джона оценки 'B' по физкультуре.
int rows = statement.executeUpdate(
  "UPDATE STUDENTTABLE SET GRADE_SECOND_PERIOD = 'B' WHERE STUDENT_ID= '123456'");

// Создание промежуточной точки сохранения в транзакции
Savepoint savepoint1 = connection.setSavepoint("SAVEPOINT_1");

// Добавление в запись Джейн оценки 'C' по биохимии.
int rows = statement.executeUpdate(
   "UPDATE STUDENTTABLE SET GRADE_SECOND_PERIOD = 'C' WHERE STUDENT_ID= '987654'");

// Обнаружена ошибка; выполняется откат записи Джейн, но не записи Джона.
// Откат транзакции к точке сохранения 1. Изменение в записи Джейн удалено,
// а изменение в записи Джона сохранено.
```

```
connection.rollback(savepoint1);

// Фиксация транзакции; в базу данных занесена только оценка Джона.
connection.commit();
```

Рекомендации и ограничения

При работе с точками сохранения следует учитывать следующие рекомендации и ограничения:

Рекомендации

IBM Toolbox for Java follows database rules regarding how rollbacks affect cursors and retained locks. Например, если в параметрах соединения указано, что после обычного отката курсоры остаются открытыми, то то же самое произойдет и после отката к точке сохранения. In other words, when a rollback request happens involving savepoints, IBM Toolbox for Java does not move or close the cursor when the underlying database does not support this.

Во время выполнения отката до точки сохранения отменяются только те действия, которые были выполнены с момента отката до точки сохранения. Все действия, выполненные до точки сохранения, остаются в силе. Как показано в предыдущем примере, при фиксации транзакции могут быть сохранены только те действия, которые были выполнены до точки сохранения.

После фиксации транзакции или отката всей транзакции созданные точки сохранения разблокируются и становятся недействительными. При необходимости точки сохранения можно разблокировать с помощью метода Connection.releaseSavepoint().

Ограничения

При работе с точками сохранения следует учитывать следующие ограничения:

- Имена точек сохранения должны быть уникальными.
- Имя точки сохранения можно повторно использовать только после разблокирования, фиксации или отката точки сохранения.
- Для применения точек сохранения необходимо выключить функцию автоматической фиксации. Это можно сделать с помощью метода `Connection.setAutoCommit(false)`. Включение функции автоматической фиксации во время работы с точками сохранения приведет к возникновению исключительной ситуации.
- Точки сохранения нельзя использовать с соединениями XA. При использовании точек сохранения и соединений XA возникает исключительная ситуация.
- Your server must be running i5/OS Version 5 Release 2 or later. Using savepoints when connecting (or already connected) to a server running V5R1 or earlier version of i5/OS throws an exception.

  AS400JDBCSavePoint Javadoc

Запуск операторов SQL с помощью объектов Statement:

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

Класс PreparedStatement является дочерним по отношению к классу Statement и родительским по отношению к классу CallableStatement. Для запуска запросов SQL применяются следующие объекты Statement:

- “Интерфейс Statement” на стр. 88: Выполняет простой оператор SQL без параметров.
- “Интерфейс PreparedStatement” на стр. 81 - Выполняет подготовленный оператор SQL с входными параметрами или без них.
- “Интерфейс CallableStatement” на стр. 63 - Выполняет вызов процедуры, хранящейся в базе данных. В CallableStatement можно задать параметры IN, OUT и INOUT.

Объект Statement позволяет объединить несколько команд SQL в одну группу и передать их на обработку в базу данных в пакетном режиме. Выполнение группы операторов в пакетном режиме обычно занимает меньше времени, чем выполнение операторов по отдельности, что позволяет повысить производительность. Дополнительная информация о поддержке пакетного режима приведена в разделе Изменения, внесенные в поддержку JDBC.

Перед запуском пакетного обновления рекомендуется выключить опцию автоматической фиксации. В этом случае программа будет самостоятельно решать, нужно ли фиксировать транзакцию, если возникла ошибка и была выполнена только часть команд. В JDBC 2.0 и более поздних спецификациях JDBC объект Statement хранит список команд, которые должны быть обработаны в пакетном режиме. Метод executeBatch() выполняет команды в том порядке, в котором они перечислены в списке.

Ниже перечислены некоторые действия, которые можно выполнить с помощью методов класса AS400JDBCStatement:

- Execute different kinds of statements
- Получать значения различных параметров объекта Statement, включая следующие:
  - The connection
  - Any auto-generated keys created as a result of executing the Statement
  - The fetch size and fetch direction
  - The maximum field size and maximum row limit
  - The current result set, the next result set, the type of result set, the result set concurrency, and the result set cursor holdability
- Add an SQL statement to the current batch
- Run the current batch of SQL statements

Интерфейс Statement

Для создания объекта Statement предназначен метод Connection.createStatement().

Ниже приведен пример работы с объектом Statement.

```
                    // Подключение к серверу.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");

                    // Создание объекта Statement.
Statement s = c.createStatement();

                    // Запуск оператора SQL, создающего
                    // таблицу в базе данных.
s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

                    // Запуск оператора SQL, вставляющего
                    // запись в таблицу.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('DAVE', 123)");

                    // Запуск оператора SQL, вставляющего
                    // запись в таблицу.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('CINDY', 456)");

                    // Запуск запроса SQL на выбор данных из таблицы.
ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

                    // Закрытие объектов Statement и
                    // Connection.
s.close();
c.close();
```

AS400JDBCStatement Javadoc

Управление распределенными транзакциями XA JDBC:

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

Обычно классы управления распределенными транзакциями XA применяются диспетчером транзакций, который не входит в состав драйвера JDBC. Интерфейсы управления распределенными транзакциями входят в состав дополнительного пакета JDBC 2.0 и API Транзакции Java (JTA). Эти пакеты можно загрузить с Web-сайта фирмы Sun в виде файлов jar. Интерфейсы управления распределенными транзакциями также поддерживаются API JDBC 3.0, встроенным в платформу Java 2, Standard Edition, версии 1.4.

Дополнительная информация приведена на Web-сайтах Sun в разделах JDBC  и JTA  .

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

- AS400JDBCXADataSource - Фабрика классов для объекта AS400JDBCXAConnection.Это подкласс класса AS400JDBCDataSource.
- Объект соединения из пула AS400JDBCXACConnection , предоставляющий точки прерывания для управления пулом соединений и ресурсами XA.
- AS400JDBCXAResource - диспетчер ресурсов для управления транзакциями XA.

Примечание: До версии V5R3, сервер базы данных применял API XA для блокировок в области заданий (модель XA). В V5R3 и последующих выпусках сервер базы данных применяет API XA для блокировок в области транзакций (модель NTS) для всех функций MTS. С дополнительной информацией о различии этих API можно ознакомиться в разделе API XA.

Пример: Применение классов XA

Ниже приведен пример работы с классами XA. Обратите внимание, что для работы с другими источниками данных потребуется значительно дополнить приведенный пример кода. Такой код обычно содержит диспетчер транзакций.

```
// Создать источник XA для установления соединения XA.
AS400JDBCXADataSource xaDataSource = new AS400JDBCXADataSource("myAS400");
xaDataSource.setUser("myUser");
xaDataSource.setPassword("myPasswd");

// Получение соединения XAConnection и связанного с ним ресурса XAResource.
// Эти объекты необходимы для работы с диспетчером ресурсов.
XAConnection xaConnection = xaDataSource.getXAConnection();
XAResource xaResource = xaConnection.getXAResource();

// Создание нового Xid (зависит от диспетчера транзакций).
Xid xid = ...;

// Начать транзакцию.
xaResource.start(xid, XAResource.TMNOFLAGS);

// ...Выполнение операций над базой данных...

// Завершить транзакцию.
xaResource.end(xid, XAResource.TMSUCCESS);

// Подготовка к фиксации.
xaResource.prepare(xid);

// Зафиксировать транзакцию.
xaResource.commit(xid, false);
```

```
    // Закрыть соединение XA. При этом будет неявно
// закрыт ресурс XA.
    xaConnection.close();
```

Классы заданий

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

Классы заданий позволяют просмотреть и изменить следующую информацию о задании:

- Дату и время
- Очередь задания
- Идентификаторы языка
- Ведение протокола сообщений
- Очередь вывода
- Информацию о принтере

Примеры

Ниже приведены примеры применения классов Job, JobList и JobLog. В первом примере показано, как можно использовать кэш при работе с классом Job. При выборе ссылок на другие примеры будет показан код программ.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Пример: Применение кэша при получении и установке значений

```
R   try {
R
R       // Создание объекта AS400.
R       AS400 as400 = new AS400("systemName");
R
R       // Создание объекта Job
R       Job job = new Job(as400,"QDEV002");
R
R       // Получение информации о задании
R       System.out.println("Владелец задания:" + job.getUser());
R       System.out.println("Использование CPU:" + job.getCPUUsed();
R       System.out.println("Дата запуска задания : " + job.getJobEnterSystemDate());
R
R       // Разрешение занесения изменений в кэш
R       job.setCacheChanges(true);
R
R       // Изменения будут сохраняться в кэше.
R       job.setRunPriority(66);
R       job.setDateFormat("*YMD");
R
R       // Commit changes. This will change the value on the system.
R       job.commitChanges();
R
R       // Передача информации о задании в систему напрямую, минуя кэш.
R       job.setCacheChanges(false);
R       job.setRunPriority(60);
R   } catch (Exception e)
R   {
R       System.out.println(quot;error :" + e)
R   }
```

В приведенных ниже примерах показано, как можно просмотреть список заданий конкретного пользователя, просмотреть список заданий с данными о состоянии заданий и просмотреть сообщения в протоколе заданий:

“Пример: Составление списка заданий с помощью объекта JobList” на стр. 528

“Пример: Получение списка заданий с помощью объекта JobList” на стр. 530

“Пример: Просмотр сообщений протокола заданий с помощью объекта JobLog” на стр. 534

Access package Javadoc

Класс Job:

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

В частности, с их помощью можно получить информацию о следующих объектах:

- Очереди заданий
- Очереди вывода
- Message logging
- Printer device
- Country or region identifier
- Формат даты

The job class also allows the ability to change a single value at a time, or cache several changes using the setCacheChanges(true) method and committing the changes using the commitChanges() method. Если кэш не применяется, то фиксировать изменения не нужно.

Пример

Пример программы приведен в справочной документации Java для класса Job. The example shows how to set and get values to and from the cache in order to set the run priority with the setRunPriority() method and set the date format with the setDateFormat() method.

Job Javadoc

Класс JobList:

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

С помощью класса JobList можно получить следующую информацию:

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

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

Пример: Применение метода JobList

Ниже приведен пример получения списка активных заданий системы:

```
R                       // Создание объекта AS400. Просмотр заданий
R // jobs on this server.
R      AS400 sys = new AS400("mySystem.myCompany.com");
R
R                       // Создание объекта, представляющего список заданий.
R      JobList jobList = new JobList(sys);
R
R                       // Получение списка активных заданий.
R      Enumeration list = jobList.getJobs();
R
R                       // Печать информации об активных
```

```
R                       // заданиях.
R      while (list.hasMoreElements())
R      {
R          Job j = (Job) list.nextElement();
R
R          System.out.println(j.getName() + "." +
R j.getUser() + "." +
R j.getNumber());
R      }
```

Job Javadoc

Класс JobLog:

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

Пример: Применение метода JobLog

Ниже приведен пример, в котором печатаются все сообщения из протокола задания текущего пользователя:

```
R                       // ... Предполагается, что объект AS/400
R                       // и объект списка заданий
R                       // уже созданы
R
R                       // Получение списка активных заданий
R                       // the server
R      Enumeration list = jobList.getJobs();
R
R                       // Выбор задания указанного
R                       // пользователя из списка.
R      while (list.hasMoreElements())
R      {
R         Job j = (Job) list.nextElement();
R
R         if (j.getUser().trim().equalsIgnoreCase(userID))
R         {
R                       // Задание текущего пользователя
R                       // найдено. Создание объекта протокола
R // для этого задания.
R            JobLog jlog = new JobLog(system, j.getName(), j.getUser(), j.getNumber());
R
R                       // Создание списка сообщений из протокола
R                       // и печать сообщений.
R            Enumeration messageList = jlog.getMessages();
R
R            while (messageList.hasMoreElements())
R            {
R                AS400Message message = (AS400Message) messageList.nextElement();
R                System.out.println(message.getText());
R            }
R
R         }
R      }
```

Job Javadoc

Классы Message

The IBM Toolbox for Java AS400Message class and its associated classes represent a message returned from a server.

Класс AS400Message

The AS400Message object allows the Java program to retrieve an i5/OS message that is generated from a previous operation (for example, from a command call). Программа на Java может получить следующую информацию из объекта сообщения:

- The System i5 library and message file that contain the message

- The message ID
- Тип сообщения
- Серьезность сообщения
- The message text
- The message help text

Ниже приведен пример использования объекта AS400Message:

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

```
                    // Создание объекта вызова команды.
CommandCall cmd = new CommandCall(sys, "myCommand");

                    // Запуск команды
cmd.run();

                    // Получение списка сообщений,
                    // выданных в ходе выполнения
                    // этой команды
AS400Message[] messageList = cmd.getMessageList();

                   // Вывод в цикле
                   // сообщений из списка
for (int i = 0; i < messageList.length; i++)
{
   System.out.println(messageList[i].getText());
}
```

Пример: Применение списков сообщений

Ниже приведены примеры применения списков сообщений с CommandCall и ProgramCall.

- “Пример: применение объекта CommandCall” на стр. 475
- “Пример: Применение объекта ProgramCall” на стр. 546

Класс QueuedMessage

The QueuedMessage class extends the AS400Message class.

The QueuedMessage class accesses information about a message on an System i5 message queue. Этот класс позволяет получать следующую информацию в программах на Java:

- Information about where a message originated, such as program, job number, and user.
- The message queue
- The message key
- The message reply status

Ниже приведен пример, в котором печатаются все сообщения из очереди сообщений текущего пользователя:

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

```
R                    // The message queue is on this system.
R       AS400 sys = new AS400(mySystem.myCompany.com);
R
R                    // Создание объекта очереди сообщений.
R                    // Этот объект представляет очередь
R                    // текущего пользователя.
R       MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);
R
R                    // Получение списка сообщений из
R                    // очереди пользователя.
```

```
R       Enumeration e = queue.getMessages();
R
R                       // Печать всех сообщений из очереди.
R       while (e.hasMoreElements())
R       {
R          QueuedMessage msg = e.getNextElement();
R          System.out.println(msg.getText());
R       }
```

Класс MessageFile

R The MessageFile class allows you to receive a message from an System i5 message file. Он возвращает объект
R AS400Message, содержащий сообщение. С помощью класса MessageFile можно выполнить следующие
R действия:

- Return a message object that contains the message
- Return a message object that contains substitution text in the message

Ниже приведен пример получения и печати сообщения:

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

```
AS400 system = new AS400("mysystem.mycompany.com");
MessageFile messageFile = new MessageFile(system);
messageFile.setPath("/QSYS.LIB/QCPFMSG.MSGF");
AS400Message message = messageFile.getMessage("CPD0170");
System.out.println(message.getText());
```

Класс MessageQueue

R The MessageQueue class allows a Java program to interact with an System i5 message queue.

Класс MessageQueue служит контейнером для класса QueuedMessage. The getMessages() method, in particular, returns a list of QueuedMessage objects. С помощью класса MessageQueue можно выполнить следующие действия:

- Set message queue attributes
- Get information about a message queue
- Receive messages from a message queue
- Send messages to a message queue
- Reply to messages

Ниже приведен пример получения списка сообщений из очереди текущего пользователя:

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

```
R                       // The message queue is on this system.
R       AS400 sys = new AS400(mySystem.myCompany.com);
R
R                       // Создание объекта очереди сообщений.
R                       // Этот объект представляет очередь
R                       // текущего пользователя.
R       MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);
R
R                       // Получение списка сообщений из
R                       // очереди пользователя.
R       Enumeration e = queue.getMessages();
R
R                       // Печать всех сообщений из очереди.
R       while (e.hasMoreElements())
```

```
R       {
R          QueuedMessage msg = e.getNextElement();
R          System.out.println(msg.getText());
R       }
```

AS400Message Javadoc

QueuedMessage Javadoc

Access package summary

MessageFile Javadoc

MessageQueue Javadoc

NetServer

NetServer устарел и заменен классом ISeriesNetServer.

The NetServer class represents the NetServer service on the server. Объекты NetServer применяются для просмотра и изменения информации о состоянии и конфигурации службы NetServer.

Например, с помощью класса NetServer можно выполнить следующие операции:

- Start or stop the NetServer
- Get a list of all current file shares and print shares
- Get a list of all current sessions
- Query and change attribute values (using methods inherited from ChangeableResource)

Примечание: Для применения класса NetServer необходим пользовательский профайл с правами доступа *IOSYSCFG.

The NetServer class is an extension of ChangeableResource and Resource, so it provides a collection of ″attributes″ to represent the various NetServer values and settings. You query or change the attributes in order to access or change the configuration of your NetServer. Ниже перечислены некоторые из таких атрибутов NetServer:

- Заказчик
- NAME_PENDING
- DOMAIN
- ALLOW_SYSTEM_NAME
- AUTOSTART
- CCSID
- WINS_PRIMARY_ADDRESS

Атрибуты с отложенным изменением

Many of the NetServer attributes are pending (for example, NAME_PENDING). Такие атрибуты представляют параметры NetServer, изменение которых откладывается до следующего запуска (или перезапуска) сервера NetServer.

Если есть пара связанных друг с другом атрибутов, изменение одного из которых отложено, то:

- Атрибут с отложенным изменением можно изменить, поскольку к нему разрешен доступ для чтения/записи
- Доступ к другому атрибуту разрешен только для чтения, поэтому его значение можно только получить, но не изменить

Другие классы NetServer

Другие классы для работы с NetServer позволяют получать и изменять информацию о соединениях, сеансах, общих каталогах и принтерах:

- NetServerConnection: Represents a NetServer connection
- NetServerFileShare: Represents a NetServer file server share
- NetServerPrintShare: Represents a NetServer print server share
- NetServerSession: Represents a NetServer session
- NetServerShare: Represents a NetServer share

Пример: Изменение имени NetServer с помощью объекта NetServer

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
// Create a system object to represent the server.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWD");

// Создание объекта для получения информации и внесения изменений в NetServer.
NetServer nServer = new NetServer(system);

// Задать имя NEWNAME в атрибуте с отложенным изменением.
nServer.setAttributeValue(NetServer.NAME_PENDING, "NEWNAME");

// Зафиксировать изменения. При этом изменения будут отправлены на сервер.
nServer.commitAttributeChanges();

// Имя NetServer изменится на NEWNAME при следующем запуске
// NetServer.
```

ObjectReferences class

The IBM Toolbox for Java ObjectReferences class represents the set of information about integrated file system references on an object that can be retrieved through the Retrieve Object References (QP0LROR) API.

A reference is an individual type of access or lock obtained on the object when using integrated file system interfaces. An object may have multiple references concurrently held, provided that the reference types do not conflict with one another. This class will not return information about byte range locks that may currently be held on an object.

The user must have execute (*X) data authority to each directory preceding the object whose references are to be obtained. The user must have read (*R) data authority to the object whose references are to be obtained.

For more information, see the ObjectReferences Javadoc.

Информация, связанная с данной

ObjectReferences Javadoc

Retrieve Object References (QP0LROR) API

Классы прав доступа

The IBM Toolbox for Java permission classes allow you to get and set object authority information. Иногда такую информацию называют просто правами доступа. Класс Permission предназначен для получения и изменения информации о правах доступа к данному объекту для множества пользователей, а класс UserPermission - только для одного пользователя.

Класс Permission

The Permission class allows you to retrieve and change object authority information. Данный класс позволяет получать информацию о пользователях, которым предоставлены права доступа к конкретному объекту. Объект Permission позволяет программе на Java хранить в кэше измененную информацию о правах доступа до тех пор, пока не будет вызван метод commit(). При вызове метода commit() все изменения, сделанные к данному моменту, фиксируются на сервере. Ниже перечислены некоторые функции, реализованные в классе Permission:

- addAuthorizedUser(): Adds an authorized user.

- commit(): Commits the permission changes to the server.
- getAuthorizationList(): Returns the authorization list of the object.
- getAuthorizedUsers(): Returns an enumeration of authorized users.
- getOwner(): Returns the name of the object owner.
- getSensitivityLevel(): Returns the sensitivity level of the object.
- getType(): Returns the object authority type (QDLO, QSYS, or Root).
- getUserPermission(): Returns the permission of a specific user to the object.
- getUserPermissions(): Returns an enumeration of permissions of the users to the object.
- setAuthorizationList(): Sets the authorization list of the object.
- setSensitivityLevel(): Sets the sensitivity level of the object.

Пример: Применение прав доступа

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведен пример создания прав доступа и добавления пользователя с правами доступа к объекту.

```
// Создание объекта AS400
AS400 as400 = new AS400();

// Создание объекта Permission и его передача в AS400
Permission myPermission = new Permission(as400, "QSYS.LIB/myLib.LIB");

// Добавление пользователя с правами доступа к объекту
myPermission.addAuthorizedUser("User1");
```

Класс UserPermission

The UserPermission class represents the authority of a single, specific user. Класс UserPermission включает три подкласса для работы с разными типами объектов:

- DLOPermission
- Класс QSYSPermission
- Класс RootPermission

Класс UserPermission позволяет выполняет следующие операции:

- Determine if the user profile is a group profile
- Return the user profile name
- Indicate whether the user has authority
- Set the authority of authorization list management

Пример: Применение класса UserPermission

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Приведенный ниже пример содержит информацию о том, как можно получить и напечатать список пользователей и групп с правами доступа к объекту.

```
// Создание объекта системы.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Создание прав доступа к объекту в системе (например, к библиотеке).
Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");
```

```
// Получение списка пользователей/групп, которым предоставлены
// права доступа к данному объекту.
Enumeration enum = objectInQSYS.getUserPermissions();
while (enum.hasMoreElements())
{
  // Поочередная печать имен профайлов пользователей/групп.
  UserPermission userPerm = (UserPermission)enum.nextElement();
  System.out.println(userPerm.getUserID());
}
```

Permission Javadoc

UserPermission Javadoc

Класс DLOPermission:

The DLOPermission class is a subclass of UserPermission. DLOPermission позволяет просматривать и задавать права доступа пользователя к объектам библиотеки документов (DLO).DLOs are stored in QDLS.

Каждому пользователю назначено одно из следующих значений, определяющих права доступа:

Права доступа	**Описание**
*ALL	Пользователь может выполнять все операции, за исключением тех, которые контролируются списками прав доступа.
*AUTL	Права доступа к документу определяются согласно списку прав доступа.
*CHANGE	Пользователь может изменять объект и выполнять основные действия.
*EXCLUDE	Пользователю запрещен доступ к объекту.
*USE	Пользователю предоставлены операционные права доступа к объекту, права на чтение и права на выполнение.

Для определения и изменения прав доступа пользователя применяются следующие методы:

- Метод getDataAuthority() позволяет просмотреть права доступа пользователя
- Метод setDataAuthority() позволяет задать права доступа пользователя

После изменения прав доступа необходимо вызвать метод commit() класса Permissions для сохранения изменений на сервере.

R For more information about permissions and authorities, see the Security reference topic.

Пример: Применение класса DLOPermission

Ниже приведен пример получения и печати прав доступа DLO и пользовательских профайлов для этих прав доступа.

```
// Создание объекта системы.

AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");
// Создание объекта для прав доступа к объекту DLO.
Permission objectInQDLS = new Permission(sys, "/QDLS/MyFolder");

// Вывод полного имени объекта и получение прав доступа к нему.
System.out.println("Права доступа к объекту " + objectInQDLS.getObjectPath() + " :");
Enumeration enum = objectInQDLS.getUserPermissions();
while (enum.hasMoreElements())
{
  // Для каждого элемента списка прав доступа выводится имя пользовательского профайла
```

```
    // и предоставленные ему права доступа к объекту.
    DLOPermission dloPerm = (DLOPermission)enum.nextElement();
    System.out.println(dloPerm.getUserID() + ": " + dloPerm.getDataAuthority());
  }
```

Информация, связанная с данной

DLOPermission Javadoc

Класс QSYSPermission:

QSYSPermission is a subclass of the UserPermission class. QSYSPermission allows you to display and set the permission a user has for an object in the traditional System i library structure stored in QSYS.LIB. Для объекта из QSYS.LIB можно задать единые права доступа к объекту и к данным (указав системное значение прав доступа), либо отдельные права доступа к объекту и к данным.

В приведенной ниже таблице описаны допустимые системные значения прав доступа:

Системное значение прав доступа	Описание
*ALL	Пользователь может выполнять все операции, за исключением тех, которые контролируются списками прав доступа.
*AUTL	Права доступа к документу определяются согласно списку прав доступа.
*CHANGE	Пользователь может изменять объект и выполнять основные действия.
*EXCLUDE	Пользователю запрещен доступ к объекту.
*USE	Пользователю предоставлены операционные права доступа к объекту, права на чтение и права на выполнение.

Все системные значения прав доступа представляют собой комбинацию прав доступа к объекту и прав доступа к данным. В следующей таблице указаны права доступа к объекту и данным, соответствующие различным системным значениям прав доступа:

Таблица 1. **Д** *относится к тем правам доступа, которые можно предоставить. н относится к правам доступа, которые нельзя предоставить.*

Системные права доступа	Права доступа к объекту					Права доступа к данным				
	Операционные	На управление	К существованию	На изменение	На обращение	На чтение	На добавление	На обновление	На удаление	На выполнение
All	**Д**	**Д**	**Д**	**Д**	**Д**	**Д**	**Д**	**Д**	**Д**	**Д**
Change	**Д**	н	н	н	н	**Д**	**Д**	**Д**	**Д**	**Д**
Exclude	н	н	н	н	н	н	н	н	н	н
Use	**Д**	н	н	н	н	**Д**	н	н	н	**Д**
Autl	Доступно только для пользователя (*PUBLIC) и указанного списка прав доступа, определяющего отдельные права доступа к объектам и к данным.									

При назначение системных прав доступа автоматически устанавливаются соответствующие отдельные права доступа. Аналогично, при изменении отдельных прав доступа переопределяются заданные ранее системные права. Если сочетание отдельных прав доступа к объекту и к данным не соответствует ни одному из вышеперечисленных системных значений прав доступа, то устанавливается значение ″Пользовательские″.

Use the getObjectAuthority() method to display the current system-defined authority. Use the setObjectAuthority() method to set the current system-defined authority using a single value.

Для предоставления или аннулирования отдельных прав доступа к объекту предназначены следующие методы:

- setAlter()
- setExistence()
- setManagement()
- setOperational()
- setReference()

Для предоставления или аннулирования отдельных прав доступа к данным предназначены следующие методы:

- setAdd()
- setDelete()
- setExecute()
- setRead()
- setUpdate()

R For more information about the different authorities, see the Security reference topic. For information about using CL
R commands to grant and edit object authorities, see the CL commands Grant Object Authority (GRTOBJAUT) and Edit
R Object Authority (EDTOBJAUT).

Пример

В данном примере показано, как получить и напечатать информацию о правах доступа к объекту QSYS.

```
// Создание объекта системы.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Предоставление прав доступа к объекту QSYS.
Permission objectInQSYS = new Permission(sys, "/QSYS.LIB/FRED.LIB");

// Вывод полного имени объекта и получение прав доступа к нему.
System.out.println("Установлены следующие права доступа к объекту "+
                   objectInQSYS.getObjectPath()+":");
Enumeration enum = objectInQSYS.getUserPermissions();
while (enum.hasMoreElements())
{
  // Для каждого элемента списка прав доступа выводится имя пользовательского профайла
  // и права доступа пользователя к объекту.
  QSYSPermission qsysPerm = (QSYSPermission)enum.nextElement();
  System.out.println(qsysPerm.getUserID()+": "+qsysPerm.getObjectAuthority());
}
```

QSYSPermission Javadoc

“Класс UserPermission” на стр. 97

Класс RootPermission:

The RootPermission class represents a user’s authority to objects contained in the root directory structure. К объектам RootPermission относятся те, которые не содержатся ни в QSYS.LIB, ни в QDLS.

RootPermission is a subclass of the UserPermission class. Класс RootPermission позволяет предоставлять пользователю права доступа к объекту структуры корневого каталога, а также получать информацию о таких правах доступа.

Для объекта структуры корневого каталога можно задавать права доступа к объекту и права доступа к данным по отдельности. Список значений прав доступа к данным приведен в следующей таблице. Use the getDataAuthority() method to to display the current values and the setDataAuthority() method to set the data authority.

В следующей таблице перечислены и описаны допустимые значения прав доступа к данным:

Права доступа к данным	Описание
*none	У пользователя нет прав доступа к объекту.
*RWX	Пользователю предоставлены права на чтение, добавление, обновление, удаление и выполнение.
*RW	Пользователю предоставлены права на чтение, добавление и удаление.
*RX	Пользователю предоставлены права на чтение и выполнение.
*WX	Пользователю предоставлены права на добавление, обновление, удаление и выполнение.
*R	Пользователю предоставлены права на чтение.
*W	Пользователю предоставлены права на добавление, обновление и удаление.
*X	Пользователю предоставлены права на выполнение.
*EXCLUDE	Пользователю запрещен доступ к объекту.
*AUTL	Общие права доступа к объекту задаются с помощью списка прав доступа.

Можно устанавливать следующие права доступа к объекту: права на изменение объекта, на существование объекта, на управление объектом или на обращение к объекту. You can use the setAlter(), setExistence(), setManagement(), or setReference() methods to set the values on or off.

After setting either the data authority or the object authority of an object, it is important that you use the commit() method from the Permissions class to send the changes to the server.

For more information about the different authorities, see the Security reference topic.

Пример

В данном примере показано, как получить и напечатать информацию о правах доступа к объекту корневой файловой системы.

```
// Создание объекта системы.
AS400 sys = new AS400("MYAS400", "USERID", "PASSWORD");

// Создайте права доступа к объекту в корневой файловой системе.
Permission objectInRoot = new Permission(sys, "/fred");

// Вывод полного имени объекта и получение прав доступа к нему.
System.out.println("Права доступа к объекту " + objectInQDLS.getObjectPath() + " :");
Enumeration enum = objectInRoot.getUserPermissions();
while (enum.hasMoreElements())
{
  // Для каждого элемента списка прав доступа выводятся имя пользовательского профайла
  // и права доступа пользователя к объекту.
  RootPermission rootPerm = (RootPermission)enum.nextElement();
  System.out.println(rootPerm.getUserID()+": "+rootPerm.getDataAuthority());
}
```

Информация, связанная с данной

RootPermission Javadoc

Классы печати

Объекты печати - это буферные файлы, очереди вывода, принтеры, файлы принтеров, задания загрузчика, а также ресурсы Advanced Function Printing (AFP), содержащие шрифты, определения форм, перекрытия, определения страниц и сегменты страниц.

The IBM Toolbox for Java classes for print objects are organized on a base class, PrintObject, and on a subclass for each of the six types of print objects. Базовый класс содержит методы и атрибуты, общие для всех объектов печати сервера. Подклассы содержат методы и атрибуты, относящиеся к отдельному подтипу.

Примеры

- Раздел Пример: Создание буферных файлов содержит информацию о создании на сервере буферного файла, содержащего поток входных данных
- Раздел Пример: Создание буферных файлов SCS содержит информацию о создании потока данных SCS с помощью класса SCS3812Writer и о сохранении потока данных в буферном файле на сервере
- Раздел Пример: Чтение буферных файлов содержит информацию о чтении буферных файлов на сервере с помощью класса PrintObjectInputStream
- Раздел Пример: Чтение и преобразование буферных файлов посвящен чтению и преобразованию данных в буферных файлах с помощью классов PrintObjectPageInputStream и PrintObjectTransformedInputStream
- Раздел Пример: Копирование буферного файла содержит информацию о копировании буферного файла в очередь сообщений, содержащую файл, который необходимо скопировать.
- Раздел Пример: Просмотр списка буферных файлов в асинхронном режиме (с помощью обработчиков) содержит информацию о том, как можно просмотреть список всех буферных файлов системы в асинхронном режиме и информацию о получении уведомлений с помощью интерфейса PrintObjectListListener во время создания этого списка
- Раздел Пример: Просмотр списка буферных файлов в асинхронном режиме (без обработчиков) содержит информацию о просмотре списка всех буферных файлов системы *без* применения интерфейса PrintObjectListListener
- Раздел Пример: Просмотр списка буферных файлов в синхронном режиме содержит информацию о просмотре списка всех буферных файлов системы в синхронном режиме

  Информация, связанная с данной

  PrintObject Javadoc

Получение списка объектов печати:

You can use the IBM Toolbox for Java PrintObjectList class and its subclasses to work with lists of print objects. Print objects include spooled files, output queues, printers, Advanced Function Printing (AFP) resources, printer files, and writer jobs.

Каждый подкласс содержит методы, которые позволяют фильтровать список по критерию, существенному для данного конкретного типа объекта печати. For example, SpooledFileList allows you to filter a list of spooled files based on the user who created the spooled files, the output queue that the spooled files are on, the form type, or user data of the spooled files. В список будут включены только файлы, удовлетворяющие заданным критериям. Если фильтры не установлены, то будут использоваться заданные для них значения по умолчанию.

To actually retrieve the list of print objects from the server, the openSynchronously() or openAsynchronously() methods are used. Метод openSynchronously() возвращает список только после того, как из сервера будут получены все объекты. Метод openAsynchronously() возвращает данные немедленно, и во время формирования списка (происходящего в фоновом режиме) инициатор может выполнять другие действия. При асинхронном открытии списка инициатор может запускать сеанс просмотра объектов пользователем сразу после поступления информации об объекте. Поскольку в этом случае пользователь видит объекты по мере их поступления, у него может создаться впечатление, что он получает ответ быстрее, чем в случае синхронного открытия списка, но в действительности полное время ответа может быть гораздо больше, поскольку при обработке каждого объекта в списке системе приходится выполнять лишние операции.

При асинхронном открытии списка инициатор может получать уточняющую информацию непосредственно в процессе создания этого списка. Methods, such as isCompleted() and size(), indicate whether the list has finished being built or return the current size of the list. Other methods, waitForListToComplete() and waitForItem(), allow the caller to wait for the list to complete or for a particular item. Кроме вызова указанных методов класса PrintObjectList инициатор может зарегистрироваться с данным списком как обработчик событий. В этом

случае инициатор будет получать уведомления о событиях, происходящих со списком. To register or unregister for the events, the caller uses PrintObjectListListener(), and then calls addPrintObjectListListener() to register removePrintObjectListListener()or to unregister. В приведенной ниже таблице перечислены события, уведомления о которых доставляются с помощью методов класса PrintObjectList.

Событие PrintObjectList	**Когда доставляется уведомление о событии**
listClosed	При закрытии списка.
listCompleted	При окончании создания списка.
listErrorOccurred	При возникновении любой исключительной ситуации при получении списка.
listOpened	При открытии списка.
listObjectAdded	При добавлении объекта в список.

After the list has been opened and the objects in the list processed, close the list using the close() method. При этом освобождаются ресурсы, которые были выделены программе очистки памяти во время открытия списка. После закрытия списка можно изменить настройки фильтров, а затем вновь открыть список.

При создании списка объектов печати из сервера пересылаются атрибуты для каждого объекта, которые хранятся вместе с ним. These attributes can be updated using the update() method in the PrintObject class. Какие именно атрибуты будут пересылаться из сервера, зависит от типа объекта печати, включаемого в список. A default list of attributes for each type of print object that can be overridden by using the setAttributesToRetrieve() method in PrintObjectList exists. Списки атрибутов, поддерживаемых для различных типов объектов печати, приведены в разделе Получение атрибутов PrintObject.

Примеры

Ниже приведены примеры различных способов просмотра списка буферных файлов.

Раздел “Пример: Асинхронное создание списка буферных файлов (с помощью обработчиков событий)” на стр. 540 содержит информацию о том, как просмотреть список всех буферных файлов системы в асинхронном режиме, а также о том, как с помощью интерфейса PrintObjectListListener можно получать уведомления во время создания этого списка

Раздел “Пример: Асинхронное создание списка буферных файлов (без помощи обработчиков событий)” на стр. 543 содержит информацию том, как просмотреть список всех буферных файлов системы в асинхронном режиме *без* применения интерфейса PrintObjectListListener

Раздел “Пример: Создание списка буферных файлов в синхронном режиме” на стр. 545 содержит информацию о том, как просмотреть список всех буферных файлов системы в синхронном режиме

PrintObjectList Javadoc

SpooledFileList Javadoc

AFPResource Javadoc

Работа с объектами печати:

PrintObject is an abstract class. An abstract class does not allow you to create an instance of the class. Instead, you must create an instance of one of its subclasses to work with print objects.

Создавать объекты подклассов можно следующими способами:

- Если вам известна система и атрибуты объекта, создайте объект с помощью явного вызова общего конструктора.
- You can use a PrintObjectList subclass to build a list of the objects and then get at the individual objects through the list.

- Вы можете воспользоваться соответствующим методом, который создает и возвращает объект, или задать вызываемые методы. For example, the static method start() in the WriterJob class returns a WriterJob object.

Use the base class, PrintObjectjavadoc/com/ibm/as400/access/PrintObject.html#NAVBAR_TOP, and its subclasses to work with server print objects:

- OutputQueue
- Принтер
- PrinterFile
- SpooledFile
- WriterJob

PrintObject Javadoc

PrintObjectList Javadoc

OutputQueue Javadoc

Printer Javadoc

PrinterFile Javadoc

SpooledFile Javadoc

WriterJob Javadoc

Получение атрибутов PrintObject:

You can retrieve print object attributes by using the attribute ID and one of several methods from the base PrintObject class.

The methods you can use include the following:

- Use getIntegerAttribute(int attributeID) to retrieve an integer type attribute.
- Use getFloatAttribute(int attributeID) to retrieve a floating point type attribute.
- Use getStringAttribute(int attributeID) to retrieve a string type attribute.

Параметр attributeID (идентификатор атрибута) - это целая константа, обозначающая атрибут. Идентификаторы определяются как общие константы в базовом классе PrintObject. Файл PrintAttributes содержит записи для всех атрибутов. Запись состоит из описания атрибута и описания его типа (целый, с плавающей точкой или строковый). Списки атрибутов, получаемых с помощью указанных методов, можно просмотреть в следующих разделах:

- AFPResourceAttrs - атрибуты ресурсов AFP
- OutputQueueAttrs - атрибуты очередей вывода
- PrinterAttrs - атрибуты принтеров
- PrinterFileAttrs - атрибуты файлов принтеров
- SpooledFileAttrs - атрибуты буферных файлов
- WriterJobAttrs - атрибуты заданий загрузчика

В целях повышения быстродействия эти атрибуты копируются на клиент, причем копирование происходит либо при создании списка объектов, либо при первом обращении к ним, если объект создан неявно. При этом не нужно подключать объект к системе каждый раз, когда приложению требуется получить некоторый атрибут. Кроме того, это позволяет сохранять в экземпляре объекта печати Java информацию об объекте на сервере, существовавшую до его изменения. The user of the object can refresh all of the attributes by calling the update() method on the object. Кроме того, атрибуты объекта обновляются автоматически, если приложение вызывает методы, приводящие к изменению атрибутов. For example, if an output queue has a status attribute of RELEASED (getStringAttribute(ATTR_OUTQSTS); returns a string of ″RELEASED″), and the hold() method is called on the output queue, getting the status attribute after that returns HELD.

Метод setAttributes

You can use the SpooledFile setAttributes method to change the attributes of spooled files and printer file objects. Список атрибутов, которые могут задаваться таким образом, приведен в следующих разделах:

- PrinterFileAttrs - атрибуты файлов принтеров
- SpooledFileAttrs - атрибуты буферных файлов

The setAttributes method takes a PrintParameterList parameter, which is a class that is used to hold a collection of attributes IDs and their values. The list starts out empty, and the caller can add attributes to the list by using the various setParameter() methods on it.

Класс PrintParameterList

Класс PrintParameterList служит для передачи группы атрибутов в метод, который далее использует любые из них в качестве параметров. For example, you can send a spooled file using TCP (LPR) by using the SpooledFile method, sendTCP(). В объекте PrintParameterList вы можете задать все параметры команды send - как обязательные (имена удаленной системы и очереди), так и необязательные (например, те, которые определяют, удалять ли буферный файл после отправки). Список обязательных и необязательных атрибутов приводится в описании каждого метода. Метод PrintParameterList setParameter() не проверяет, какие атрибуты и какие значения вы задали, - он лишь содержит значения, которые передаются данному методу. В общем случае, лишние атрибуты, заданные в PrintParameterList, игнорируются, а проверку допустимости значений используемых атрибутов выполняет сервер.

PrintObject Javadoc

SpooledFile Javadoc

PrintParameterList Javadoc

Атрибуты ресурсов AFP:

This topic lists the attributes that can be retrieved and set for an AFP resource.

Получение атрибутов

С помощью методов getIntegerAttribute(), getStringAttribute() или getFloatAttribute() можно получить следующие атрибуты ресурсов AFP:

- ATTR_AFP_RESOURCE - Путь к ресурсу AFP в интегрированной файловой системе
- ATTR_OBJEXTATTR - Расширенный атрибут объекта
- ATTR_DESCRIPTION - Текстовое описание
- ATTR_DATE - Дата открытия файла
- ATTR_TIME - Время открытия файла
- ATTR_NUMBYTES - Количество байтов для чтения/записи

Задание атрибутов

Установка атрибутов для ресурса AFP запрещена.

Атрибуты очереди вывода:

This topic lists the attributes available for an output queue.

Получение атрибутов

Ниже перечислены атрибуты, которые можно получить с помощью методов getIntegerAttribute(), getStringAttribute() и getFloatAttribute():

- ATTR_AUTHCHCK - Права на проверку
- ATTR_DATA_QUEUE - Имя очереди данных в IFS
- ATTR_DISPLAYANY - Показать все файлы
- ATTR_JOBSEPRATR - Разделители заданий
- ATTR_NUMFILES - Число файлов
- ATTR_NUMWRITERS - Число приемников для очереди данных
- ATTR_OPCNTRL - Управление оператором
- ATTR_ORDER - Порядок файлов в очереди
- ATTR_OUTPUT_QUEUE - Имя очереди вывода в IFS
- ATTR_OUTQSTS - Состояние очереди вывода
- ATTR_PRINTER - Принтер
- ATTR_SEPPAGE - Страница разделителя
- ATTR_DESCRIPTION - Текстовое описание
- ATTR_USRDEFOPT - Пользовательские опции
- ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS
- ATTR_USER_TRANSFORM_PROG - Имя пользовательской программы преобразования в IFS
- ATTR_USER_DRIVER_PROG - Имя пользовательского драйвера в IFS
- ATTR_WTRJOBNAME - Имя задания записи
- ATTR_WTRJOBNUM - Номер задания записи
- ATTR_WTRJOBSTS - Состояние задания записи
- ATTR_WTRJOBUSER - Имя пользователя задания записи

Указание атрибутов

Атрибуты очереди вывода изменить нельзя.

Атрибуты принтера:

Ниже перечислены атрибуты принтера, которые можно получить с помощью методов getIntegerAttribute), getStringAttribute() и getFloatAttribute():

Получение атрибутов
- ATTR_AFP - Advanced Function Printing
- ATTR_ALIGNFORMS - Выравнивание форм
- ATTR_ALWDRTPRT - Разрешить печать без буферизации
- ATTR_BTWNCPYSTS - Состояние обработки между копиями
- ATTR_BTWNFILESTS - Состояние обработки между файлами
- ATTR_CODEPAGE - Кодовая страница
- ATTR_CHANGES - Изменения
- ATTR_DEVCLASS - Класс устройства
- ATTR_DEVMODEL - Модель устройства
- ATTR_DEVTYPE - Тип устройства
- ATTR_DEVSTATUS - Состояние устройства
- ATTR_DRWRSEP - Лоток для разделительных страниц
- ATTR_ENDPNDSTS - Ожидающее состояние завершения
- ATTR_FILESEP - Разделители файлов
- ATTR_FONTID - Идентификатор шрифта

- ATTR_FORM_DEFINITION - Имя определения формы в IFS
- ATTR_FORMTYPE - Тип формы
- ATTR_FORMTYPEMSG - Сообщение типа формы
- ATTR_FORMFEED - Символ перевода страницы
- ATTR_CHAR_ID - Набор графических символов
- ATTR_HELDSTS - Состояние блокировки
- ATTR_HOLDPNDSTS - Ожидающее состояние блокировки
- ATTR_JOBUSER - Пользователь задания
- ATTR_MFGTYPE - Производитель, тип и модель устройства
- ATTR_MESSAGE_QUEUE - Имя очереди сообщений в IFS
- ATTR_ONJOBQSTS - Состояние очереди задания
- ATTR_OUTPUT_QUEUE - Имя очереди вывода в IFS
- ATTR_OVERALLSTS - Общее состояние
- ATTR_POINTSIZE - Размер в пунктах
- ATTR_PRINTER - Принтер
- ATTR_PRTDEVTYPE - Тип принтера
- ATTR_PUBINF_COLOR_SUP - Информация о публикации - Поддержка цветной печати
- ATTR_PUBINF_PPM_COLOR - Информация о публикации - Страниц в минуту (Цветная печать)
- ATTR_PUBINF_PPM - Информация о публикации - Страниц в минуту (Одноцветная печать)
- ATTR_PUBINF_DUPLEX_SUP - Информация о публикации - Поддержка двусторонней печати
- ATTR_PUBINF_LOCATION - Информация о публикации - Расположение
- ATTR_RMTLOCNAME - Имя удаленного расположения
- ATTR_SPOOLFILE - Имя буферного файла
- ATTR_SPLFNUM - Номер буферного файла
- ATTR_STARTEDBY - Запущено пользователем
- ATTR_DESCRIPTION - Текстовое описание
- ATTR_USERDATA - Пользовательские данные
- ATTR_USRDEFOPT - Пользовательские опции
- ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS
- ATTR_USER_TRANSFORM_PROG - Имя пользовательской программы преобразования в IFS
- ATTR_USER_DRIVER_PROG - Имя пользовательского драйвера в IFS
- ATTR_SCS2ASCII - Преобразование SCS в ASCII
- ATTR_WTNGDATASTS - Ожидание состояния данных
- ATTR_WTNGDEVSTS - Ожидание состояния устройства
- ATTR_WTNGMSGSTS - Ожидание состояния сообщения
- ATTR_WTRAUTOEND - Время автоматического завершения работы загрузчика
- ATTR_WTRJOBNAME - Имя задания загрузчика
- ATTR_WTRJOBSTS - Состояние задания загрузчика
- ATTR_WTRSTRTD - Загрузчик запущен
- ATTR_WRTNGSTS - Состояние записи

Указание атрибутов

Атрибуты принтера изменить нельзя.

Атрибуты файла принтера:

This topic contains a list of printer file attributes for use with IBM Toolbox for Java.

Получение атрибутов

Ниже перечислены атрибуты файла принтера, которые можно получить с помощью методов getIntegerAttribute(), getStringAttribute() и getFloatAttribute():

- ATTR_ALIGN - Выравнивание страницы
- ATTR_BKMGN_ACR - Горизонтальное смещение поля на обратной стороне
- ATTR_BKMGN_DWN - Вертикальное смещение поля на обратной стороне
- ATTR_BACK_OVERLAY - Имя перекрытия на обратной стороне в IFS
- ATTR_BKOVL_DWN - Вертикальное смещение перекрытия на обратной стороне
- ATTR_BKOVL_ACR - Горизонтальное смещение перекрытия на обратной стороне
- ATTR_CPI - Символов на дюйм
- ATTR_CODEDFNTLIB - Имя библиотеки кодированных шрифтов
- ATTR_CODEPAGE - Кодовая страница
- ATTR_CODEDFNT - Имя кодированного шрифта
- ATTR_CONTROLCHAR - Управляющий символ
- ATTR_CONVERT_LINEDATA - Преобразование строковых данных
- ATTR_COPIES - Копии
- ATTR_CORNER_STAPLE - Угол скрепления
- ATTR_DBCSDATA - Пользовательские данные DBCS
- ATTR_DBCSEXTENSN - Символы расширения DBCS
- ATTR_DBCSROTATE - Поворот символов DBCS
- ATTR_DBCSCPI - Символов DBCS на дюйм
- ATTR_DBCSSISO - Отступы SO/SI DBCS
- ATTR_DFR_WRITE - Отложенная запись
- ATTR_PAGRTT - Угол поворота страницы
- ATTR_EDGESTITCH_NUMSTAPLES - Число скрепок бокового скрепления
- ATTR_EDGESTITCH_REF - Базовый край бокового скрепления
- ATTR_EDGESTITCH_REFOFF - Базовый край бокового скрепления
- ATTR_ENDPAGE - Конечная страница
- ATTR_FILESEP - Разделители файлов
- ATTR_FOLDREC - Записи папки
- ATTR_FONTID - Идентификатор шрифта
- ATTR_FORM_DEFINITION - Имя определения формы в IFS
- ATTR_FORMFEED - Символ перевода страницы
- ATTR_FORMTYPE - Тип формы
- ATTR_FTMGN_ACR - Горизонтальное смещение поля на лицевой стороне
- ATTR_FTMGN_DWN - Вертикальное смещение поля на лицевой стороне
- ATTR_FRONT_OVERLAY - Имя перекрытия на лицевой стороне в IFS
- ATTR_FTOVL_ACR - Горизонтальное смещение перекрытия на лицевой стороне
- ATTR_FTOVL_DWN - Вертикальное смещение перекрытия на лицевой стороне
- ATTR_CHAR_ID - Набор графических символов
- ATTR_JUSTIFY - Аппаратное выравнивание
- ATTR_HOLD - Блокировка буферного файла
- ATTR_LPI - Строк на дюйм

- ATTR_MAXRCDS - Максимальное число буферизованных записей вывода
- ATTR_OUTPTY - Приоритет вывода
- ATTR_OUTPUT_QUEUE - Имя очереди вывода в IFS
- ATTR_OVERFLOW - Номер строки переполнения
- ATTR_PAGE_DEFINITION - Имя определения страницы в IFS
- ATTR_PAGELEN - Размер страницы
- ATTR_MEASMETHOD - Способ измерения
- ATTR_PAGEWIDTH - Ширина страницы
- ATTR_MULTIUP - Страниц на каждой стороне
- ATTR_POINTSIZE - Размер в пунктах
- ATTR_FIDELITY - Точность печати
- ATTR_DUPLEX - Двусторонняя печать
- ATTR_PRTQUALITY - Качество печати
- ATTR_PRTTEXT - Текст для печати
- ATTR_PRINTER - Принтер
- ATTR_PRTDEVTYPE - Тип принтера
- ATTR_RPLUNPRT - Заменять непечатаемые символы
- ATTR_RPLCHAR - Символ замены
- ATTR_SADDLESTITCH_NUMSTAPLES - Число скрепок скрепления посередине
- ATTR_SADDLESTITCH_REF - Базовый край скрепления посередине
- ATTR_SAVE -Сохранение буферного файла
- ATTR_SRCDRWR - Исходный лоток
- ATTR_SPOOL - Буферизация данных
- ATTR_SCHEDULE - Расписание буферизованного вывода
- ATTR_STARTPAGE - Начальная страница
- ATTR_DESCRIPTION - Текстовое описание
- ATTR_UNITOFMEAS - Единица измерения
- ATTR_USERDATA - Пользовательские данные
- ATTR_USRDEFDATA - Данные, определенные пользователем
- ATTR_USRDEFOPT - Пользовательские опции
- ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS

Указание атрибутов

Ниже перечислены атрибуты файла принтера, которые можно задать с помощью метода setAttributes():
- ATTR_ALIGN - Выравнивание страницы
- ATTR_BKMGN_ACR - Горизонтальное смещение поля на обратной стороне
- ATTR_BKMGN_DWN - Вертикальное смещение поля на обратной стороне
- ATTR_BACK_OVERLAY - Имя перекрытия на обратной стороне в IFS
- ATTR_BKOVL_DWN - Вертикальное смещение перекрытия на обратной стороне
- ATTR_BKOVL_ACR - Горизонтальное смещение перекрытия на обратной стороне
- ATTR_CPI - Символов на дюйм
- ATTR_CODEDFNTLIB - Имя библиотеки кодированных шрифтов
- ATTR_CODEPAGE - Кодовая страница
- ATTR_CODEDFNT - Имя кодированного шрифта
- ATTR_CONTROLCHAR - Управляющий символ

- ATTR_CONVERT_LINEDATA - Преобразование строковых данных
- ATTR_COPIES - Копии
- ATTR_CORNER_STAPLE - Угол скрепления
- ATTR_DBCSDATA - Пользовательские данные DBCS
- ATTR_DBCSEXTENSN - Символы расширения DBCS
- ATTR_DBCSROTATE - Поворот символов DBCS
- ATTR_DBCSCPI - Символов DBCS на дюйм
- ATTR_DBCSSISO - Отступы SO/SI DBCS
- ATTR_DFR_WRITE - Отложенная запись
- ATTR_PAGRTT - Угол поворота страницы
- ATTR_EDGESTITCH_NUMSTAPLES - Число скрепок бокового скрепления
- ATTR_EDGESTITCH_REF - Базовый край бокового скрепления
- ATTR_EDGESTITCH_REFOFF - Базовый край бокового скрепления
- ATTR_ENDPAGE - Конечная страница
- ATTR_FILESEP - Разделители файлов
- ATTR_FOLDREC - Записи папки
- ATTR_FONTID - Идентификатор шрифта
- ATTR_FORM_DEFINITION - Имя определения формы в IFS
- ATTR_FORMFEED - Символ перевода страницы
- ATTR_FORMTYPE - Тип формы
- ATTR_FTMGN_ACR - Горизонтальное смещение поля на лицевой стороне
- ATTR_FTMGN_DWN - Вертикальное смещение поля на лицевой стороне
- ATTR_FRONT_OVERLAY - Имя перекрытия на лицевой стороне в IFS
- ATTR_FTOVL_ACR - Горизонтальное смещение перекрытия на лицевой стороне
- ATTR_FTOVL_DWN - Вертикальное смещение перекрытия на лицевой стороне
- ATTR_CHAR_ID - Набор графических символов
- ATTR_JUSTIFY - Аппаратное выравнивание
- ATTR_HOLD - Блокировка буферного файла
- ATTR_LPI - Строк на дюйм
- ATTR_MAXRCDS - Максимальное число буферизованных записей вывода
- ATTR_OUTPTY - Приоритет вывода
- ATTR_OUTPUT_QUEUE - Имя очереди вывода в IFS
- ATTR_OVERFLOW - Номер строки переполнения
- ATTR_PAGE_DEFINITION - Имя определения страницы в IFS
- ATTR_PAGELEN - Размер страницы
- ATTR_MEASMETHOD - Способ измерения
- ATTR_PAGEWIDTH - Ширина страницы
- ATTR_MULTIUP - Страниц на каждой стороне
- ATTR_POINTSIZE - Размер в пунктах
- ATTR_FIDELITY - Точность печати
- ATTR_DUPLEX - Двусторонняя печать
- ATTR_PRTQUALITY - Качество печати
- ATTR_PRTTEXT - Текст для печати
- ATTR_PRINTER - Принтер
- ATTR_PRTDEVTYPE - Тип принтера

- ATTR_RPLUNPRT - Заменять непечатаемые символы
- ATTR_RPLCHAR - Символ замены
- ATTR_SADDLESTITCH_NUMSTAPLES - Число скрепок скрепления посередине
- ATTR_SADDLESTITCH_REF - Базовый край скрепления посередине
- ATTR_SAVE -Сохранение буферного файла
- ATTR_SRCDRWR - Исходный лоток
- ATTR_SPOOL - Буферизация данных
- ATTR_SCHEDULE - Расписание буферизованного вывода
- ATTR_STARTPAGE - Начальная страница
- ATTR_DESCRIPTION - Текстовое описание
- ATTR_UNITOFMEAS - Единица измерения
- ATTR_USERDATA - Пользовательские данные
- ATTR_USRDEFDATA - Данные, определенные пользователем
- ATTR_USRDEFOPT - Пользовательские опции
- ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS

Атрибуты буферного файла:

This topic lists the attributes that can be retrieved and set for a spooled file.

Получение атрибутов

Методы getIntegerAttribute(), getStringAttribute() и getFloatAttribute() позволяют получить следующие атрибуты буферного файла:

- ATTR_AFP - Advanced Function Printing
- ATTR_ALIGN - Выравнивание страницы
- ATTR_BKMGN_ACR - Горизонтальное смещение перекрытия на обратной стороне
- ATTR_BKMGN_DWN - Вертикальное смещение перекрытия на обратной стороне
- ATTR_BACK_OVERLAY - Имя перекрытия на обратной стороне в IFS
- ATTR_BKOVL_DWN - Вертикальное смещение перекрытия на обратной стороне
- ATTR_BKOVL_ACR - Горизонтальное смещение перекрытия на обратной стороне
- ATTR_CPI - Символов на дюйм
- ATTR_CODEDFNTLIB - Имя библиотеки кодированных шрифтов
- ATTR_CODEDFNT - Имя кодированного шрифта
- ATTR_CODEPAGE - Кодовая страница
- ATTR_CONTROLCHAR - Управляющий символ
- ATTR_COPIES - Копии
- ATTR_COPIESLEFT - Осталось копий
- ATTR_CORNER_STAPLE - Угол скрепления
- ATTR_CURPAGE - Текущая страница
- ATTR_DATE - Дата создания объекта
- ATTR_DATE_WTR_BEGAN_FILE - Дата начала сохранения буферного файла
- ATTR_DATE_WTR_CMPL_FILE - Дата завершения сохранения буферного файла
- ATTR_DBCSDATA - Пользовательские данные DBCS
- ATTR_DBCSEXTENSN - Символы расширения DBCS
- ATTR_DBCSROTATE - Поворот символов DBCS
- ATTR_DBCSCPI - Символов DBCS на дюйм

- ATTR_DBCSSISO - Отступы SO/SI DBCS
- ATTR_PAGRTT - Угол поворота страницы
- ATTR_EDGESTITCH_NUMSTAPLES - Число скрепок бокового скрепления
- ATTR_EDGESTITCH_REF - Базовый край бокового скрепления
- ATTR_EDGESTITCH_REFOFF - Смещение базового края бокового скрепления
- ATTR_ENDPAGE - Конечная страница
- ATTR_FILESEP - Разделители файлов
- ATTR_FOLDREC - Записи папки
- ATTR_FONTID - Идентификатор шрифта
- ATTR_FORM_DEFINITION - Имя определения формы в IFS
- ATTR_FORMFEED - Символ перевода страницы
- ATTR_FORMTYPE - Тип формы
- ATTR_FTMGN_ACR - Горизонтальное смещение поля на лицевой стороне
- ATTR_FTMGN_DWN - Вертикальное смещение поля на лицевой стороне
- ATTR_FRONTSIDE_OVERLAY - Имя перекрытия на лицевой стороне в IFS
- ATTR_FTOVL_ACR - Горизонтальное смещение перекрытия на лицевой стороне
- ATTR_FTOVL_DWN - Вертикальное смещение перекрытия на лицевой стороне
- ATTR_CHAR_ID - Набор графических символов
- ATTR_JUSTIFY - Аппаратное выравнивание
- ATTR_HOLD - Блокировка буферного файла
- ATTR_IPP_ATTR_CHARSET - Атрибуты IPP - кодировка
- ATTR_IPP_JOB_ID - ИД задания IPP
- ATTR_IPP_JOB_NAME - Имя задания IPP
- ATTR_IPP_JOB_NAME_NL - NL имени задания IPP
- ATTR_IPP_JOB_ORIGUSER - Инициатор задания IPP
- ATTR_IPP_JOB_ORIGUSER_NL - NL инициатора задания IPP
- ATTR_IPP_PRINTER_NAME - Имя принтера IPP
- ATTR_JOBNAME - Имя задания
- ATTR_JOBNUMBER - Номер задания
- ATTR_JOBUSER - Пользователь задания
- ATTR_JOB_SYSTEM - Система задания
- ATTR_LASTPAGE - Последняя напечатанная страница
- ATTR_LINESPACING - Межстрочный интервал
- ATTR_LPI - Строк на дюйм
- ATTR_MAXRCDS - Максимальное число буферизованных записей вывода
- ATTR_PAGELEN - Размер страницы
- ATTR_PAGEWIDTH - Ширина страницы
- ATTR_MEASMETHOD - Способ измерения
- ATTR_NETWORK - Идентификатор сети
- ATTR_NUMBYTES - Число байтов для чтения/записи
- ATTR_OUTPUTBIN - Принимающий лоток
- ATTR_OUTPTY - Приоритет вывода
- ATTR_OUTPUT_QUEUE - Имя очереди вывода в IFS
- ATTR_OVERFLOW - Номер строки переполнения
- ATTR_MULTIUP - Страниц на каждой стороне

- ATTR_POINTSIZE - Размер в пунктах
- ATTR_FIDELITY - Точность печати
- ATTR_DUPLEX - Двусторонняя печать
- ATTR_PRTQUALITY - Качество печати
- ATTR_PRTTEXT - Текст для печати
- ATTR_PRINTER - Принтер
- ATTR_PRTASSIGNED - Назначенный принтер
- ATTR_PRTDEVTYPE - Тип принтера
- ATTR_PRINTER_FILE - Имя файла принтера в IFS
- ATTR_RECLENGTH - Размер записи
- ATTR_REDUCE - Сокращение вывода
- ATTR_RPLUNPRT - Заменять непечатаемые символы
- ATTR_RPLCHAR - Символ замещения
- ATTR_RESTART - Повторение печати
- ATTR_SADDLESTITCH_NUMSTAPLES - Число скрепок скрепления посередине
- ATTR_SADDLESTITCH_REF - Базовый край скрепления посередине
- ATTR_SAVE - Сохранение буферного файла
- ATTR_SRCDRWR - Исходный лоток
- ATTR_SPOOLFILE - Имя буферного файла
- ATTR_SPLFNUM - Номер буферного файла
- ATTR_SPLFSTATUS - Состояние буферного файла
- ATTR_SCHEDULE - Расписание буферизованного вывода
- ATTR_STARTPAGE - Начальная страница
- ATTR_SYSTEM - Система, в которой созданы объекты
- ATTR_TIME - Время создания объекта
- ATTR_TIME_WTR_BEGAN_FILE - Время начала сохранения буферного файла
- ATTR_TIME_WTR_CMPL_FILE - Время завершения сохранения буферного файла
- ATTR_PAGES - Всего страниц
- ATTR_UNITOFMEAS - Единица измерения
- ATTR_USERCMT - Комментарий пользователя
- ATTR_USERDATA - Пользовательские данные
- ATTR_USRDEFDATA - Данные, определенные пользователем
- ATTR_USRDEFFILE - Файл, определенный пользователем
- ATTR_USRDEFOPT - Пользовательские опции
- ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS

Указание атрибутов

Ниже перечислены атрибуты буферного файла, которые можно задать с помощью метода setAttributes():
- ATTR_ALIGN - Выравнивание страницы
- ATTR_BACK_OVERLAY - Имя перекрытия на обратной стороне в IFS
- ATTR_BKOVL_DWN - Вертикальное смещение перекрытия на обратной стороне
- ATTR_BKOVL_ACR - Горизонтальное смещение перекрытия на обратной стороне
- ATTR_COPIES - Копии
- ATTR_ENDPAGE - Конечная страница
- ATTR_FILESEP - Разделители файлов

- ATTR_FORM_DEFINITION - Имя определения формы в IFS
- ATTR_FORMFEED - Символ перевода страницы
- ATTR_FORMTYPE - Тип формы
- ATTR_FRONTSIDE_OVERLAY - Имя перекрытия на лицевой стороне в IFS
- ATTR_FTOVL_ACR - Горизонтальное смещение перекрытия на лицевой стороне
- ATTR_FTOVL_DWN - Вертикальное смещение перекрытия на лицевой стороне
- ATTR_OUTPTY - Приоритет вывода
- ATTR_OUTPUT_QUEUE - Имя очереди вывода в IFS
- ATTR_MULTIUP - Страниц на каждой стороне
- ATTR_FIDELITY - Точность печати
- ATTR_DUPLEX - Двусторонняя печать
- ATTR_PRTQUALITY - Качество печати
- ATTR_PRTSEQUENCE - Порядок печати
- ATTR_PRINTER - Принтер
- ATTR_RESTART - Повторение печати
- ATTR_SAVE - Сохранение буферного файла
- ATTR_SCHEDULE - Расписание буферизованного вывода
- ATTR_STARTPAGE - Начальная страница
- ATTR_USERDATA - Пользовательские данные
- ATTR_USRDEFOPT - Пользовательские опции
- ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS

Атрибуты загрузчика:

This topic lists the attributes for writer jobs.

Получение атрибутов

Ниже перечислены атрибуты загрузчика, которые можно получить с помощью методов getIntegerAttribute(), getStringAttribute() и getFloatAttribute():

- ATTR_WTRJOBNAME - Имя задания загрузчика
- ATTR_WTRJOBNUM - Номер задания загрузчика
- ATTR_WTRJOBSTS - Состояние задания загрузчика
- ATTR_WTRJOBUSER - Имя пользователя задания загрузчика

Указание атрибутов

Атрибуты загрузчика изменять нельзя.

Атрибуты объекта печати:

This topic lists the attributes available for print objects.

- Advanced Function Printing
- AFP Resource
- Выравнивание форм
- Выравнивание страниц
- Разрешить печать без буферизации
- Права доступа

- Права на управление
- Автоматическое завершение работы загрузчика
- Вспомогательная память
- Горизонтальный отступ на обратной стороне
- Вертикальный отступ на обратной стороне
- Перекрытие на обратной стороне
- Горизонтальное смещение перекрытия на обратной стороне
- Вертикальное смещение перекрытия на обратной стороне
- Состояние между печатью копий
- Состояние между печатью файлов
- Изменения
- Число символов на дюйм
- Кодовая страница
- Имя кодированного шрифта
- Библиотека кодированного шрифта
- Управляющий символ
- Преобразование строковых данных
- Число копий
- Осталось напечатать копий
- Угол скрепления
- Текущая страница
- Формат данных
- Очередь данных
- Время открытия файла данных
- Конечная дата создания задания буферного файла
- Дата начала обработки буферного файла загрузчиком
- Дата завершения обработки буферного файла загрузчиком
- Пользовательские данные DBCS
- Символы расширенного DBCS
- Поворот символов DBCS
- Число символов DBCS на дюйм
- Пробелы вместо скобочных символов DBCS
- Запись с отсрочкой
- Угол поворота страницы
- Удаление файла после отправки
- Целевая опция
- Целевой тип
- Класс устройства
- Модель устройства
- Состояние устройства
- Тип устройства
- Просмотр всех файлов
- Лоток для разделительных страниц
- Число скрепок бокового скрепления
- Базовый край бокового скрепления

- Смещение от базового края бокового скрепления
- Состояние Завершен (ожидание)
- Последняя страница
- Размер конвертов
- Число разделительных страниц для файлов
- Перенос записей
- Идентификатор шрифта
- Определение формы
- Подача бумаги
- Тип формы
- Опция отправки сообщений о форме
- Горизонтальный отступ на лицевой стороне
- Вертикальный отступ на лицевой стороне
- Перекрытие на лицевой стороне
- Горизонтальное смещение перекрытия на лицевой стороне
- Вертикальное смещение перекрытия на лицевой стороне
- Набор графических символов
- Аппаратное выравнивание
- Состояние Блокирован
- Блокировка буферного файла
- Состояние Блокирован (ожидание)
- Конфигурация изображения
- Инициализация загрузчика
- IP-адрес
- Набор символов атрибутов IPP
- ИД задания IPP
- Имя задания IPP
- Язык имени задания IPP
- Имя пользователя, запустившего задание IPP
- Язык имени пользователя, запустившего задание IPP
- Имя принтера IPP
- Имя задания
- Номер задания
- Число разделителей заданий
- Задание системы
- Пользователь задания
- Последняя напечатанная страница
- Длина страницы
- Имя библиотеки
- Число строк на дюйм
- Межстрочный интервал
- Производитель, тип и модель
- Максимальное число заданий в списке клиента
- Максимальное число записей буферизованного вывода
- Способ измерения

- Справка по сообщению
- ИД сообщения
- Очередь сообщений
- Ответ на сообщение
- Текст сообщения
- Тип сообщения
- Серьезность сообщения
- Поддержка составного ответа
- Идентификатор сети
- Атрибуты объекта сервера сетевой печати
- Число байт в буферном файле
- Число байт для чтения/записи
- Число файлов
- Число загрузчиков, запущенных в очереди
- Расширенный атрибут объекта
- Состояние В очереди заданий
- Начальные команды
- Управляется оператором
- Порядок файлов в очереди
- Принимающий лоток
- Приоритет вывода
- Очередь вывода
- Состояние очереди вывода
- Общее состояние
- Номер строки переполнения
- Постраничный просмотр
- Приблизительное число страниц
- Определение страницы
- Номер страницы
- Число страниц на стороне
- Источник бумаги 1
- Источник бумаги 2
- Размер в пикселах
- Размер в пунктах
- Точность печати
- Печать на обеих сторонах
- Качество печати
- Порядок вывода на печать
- Текст для печати
- Printer
- Назначенный принтер
- Тип принтера
- Файл принтера
- Очередь принтера
- Информация о публикации - Поддержка цвета

- Информация о публикации - Число страниц в минуту (цветная печать)
- Информация о публикации - Число страниц в минуту (одноцветная печать)
- Информация о публикации - Поддержка двусторонней печати
- Информация о публикации - Расположение
- Имя удаленного расположения
- Длина записи
- Сокращать вывод
- Удаленная система
- Заменять непечатаемые символы
- Символ замещения
- Повторить печать
- Число скрепок скрепления посередине
- Базовый край скрепления посередине
- Сохранить буферный файл
- Смещение указателя
- Начало отсчета
- Приоритет отправки
- Разделительная страница
- Исходный лоток
- SCS буфера
- Поместить данные в буфер
- Способ идентификации при создании буферного файла
- Способ защиты для создания буферного файла
- Имя буферного файла
- Номер буферного файла
- Состояние буферного файла
- Расписание буферизованного вывода
- Запущен пользователем
- Первая страница
- Исходная система
- Текстовое описание
- Время открытия файла
- Конечное время создания задания буферного файла
- Время начала обработки загрузчиком буферного файла
- Время завершения обработки буферного файла загрузчиком
- Общее число страниц
- Преобразовать SCS в ASCII
- Единица измерения
- Комментарий пользователя
- Пользовательское описание
- Пользовательские данные
- Пользовательский файл
- Пользовательский объект
- Пользовательские опции
- Данные для пользовательского драйвера

- Пользовательский драйвер
- ИД пользователя
- Адрес пользователя
- Пользовательская программа преобразования
- Точность просмотра
- Класс VM/MVS
- Состояние Ожидание данных
- Состояние Ожидание устройства
- Состояние Ожидание сообщения
- Условие автоматического завершения работы загрузчика
- Условие завершения работы загрузчика
- Условие блокирования файла
- Ширина страницы
- Объект настройки рабочей станции
- Имя задания загрузчика
- Номер задания загрузчика
- Состояние задания загрузчика
- Имя пользователя задания загрузчика
- Загрузчик запущен
- Начальная страница загрузчика
- Состояние записи
- CCSID NPS
- Уровень NPS

Advanced Function Printing

ИД ATTR_AFP

Тип String

Описание

Указывает, применяет ли данный буферный файл внешние ресурсы AFP. Возможные значения: *YES и *NO.

Ресурс AFP

ИД ATTR_AFP_RESOURCE

Тип String

Описание

Имя внешнего ресурса AFP в интегрированной файловой системе. Полные имена объектов IFS имеют формат ″/QSYS.LIB/библиотека.LIB/ресурс.тип″, где *библиотека* - это имя библиотеки, содержащей ресурс, *ресурс* - имя ресурса, а *тип* - его тип. Допустимыми значениями для *типа* являются ″FNTRSC″, ″FORMDF″, ″OVL″, ″PAGSEG″ и ″PAGDFN″.

Выравнивание форм

ИД ATTR_ALIGNFORMS

Тип String

Описание

Время отправки сообщения о выравнивании форм. Возможные значения: *WTR, *FILE и *FIRST.

Выравнивание страниц

ИД ATTR_ALIGN

Тип String

Описание
Указывает, будет ли перед печатью данного буферного файла отправлено сообщение о выравнивании форм. Возможные значения: *YES и *NO.

Разрешить печать без буферизации

ИД ATTR_ALWDRTPRT

Тип String

Описание
Указывает, разрешает ли загрузчик принтера выделять принтер заданию, выполняющему печать без буферизации. Возможные значения: *YES и *NO.

Права доступа

ИД ATTR_AUT

Тип String

Описание
Задает права доступа, которые предоставляются пользователям без специальных прав доступа к очереди вывода. Возможные значения: *USE, *ALL, *CHANGE, *EXCLUDE и *LIBCRTAUT.

Права на управление

ИД ATTR_AUTCHK

Тип String

Описание
Указывает, какие права доступа к очереди вывода позволяют пользователю управлять всеми файлами в очереди. Возможные значения: *OWNER и *DTAAUT.

Автоматическое завершение работы загрузчика

ИД ATTR_AUTOEND

Тип String

Описание
Указывает, следует ли автоматически завершать работу загрузчика. Возможные значения: *YES и *NO.

Вспомогательная память

ИД ATTR_AUX_POOL

Тип Integer

Описание
Задает номер Пула вспомогательной памяти (ASP), в который будет помещен буферный файл. Возможные значения:

- 1: Системный ASP
- 2-32: Один из пользовательских ASP

Горизонтальный отступ на обратной стороне

ИД ATTR_BACKMGN_ACR

Тип Float

Описание
Задает отступ от левого края при печати на обратной стороне листа. Специальное значение *FRONTMGN равно -1.

Вертикальный отступ на обратной стороне

ИД ATTR_BACKMGN_DWN

Тип Float

Описание
Задает отступ от верхнего края при печати на обратной стороне листа. Специальное значение *FRONTMGN равно -1.

Перекрытие на обратной стороне

ИД ATTR_BACK_OVERLAY

Тип String

Описание
Имя перекрытия на обратной стороне в интегрированной файловой системе или специальное значение. В первом случае имя задается в формате ″/QSYS.LIB/библиотека.LIB/перекрытие.OVL″, где *библиотека* - имя библиотеки, в которой расположен ресурс, а *перекрытие* - имя перекрытия. Допустимые специальные значения: *FRONTOVL.

Горизонтальное смещение перекрытия на обратной стороне

ИД ATTR_BKOVL_ACR

Тип Float

Описание
Горизонтальное смещение от начала отсчета для печати перекрытия.

Вертикальное смещение перекрытия на обратной стороне

ИД ATTR_BKOVL_DWN

Тип Float

Описание
Вертикальное смещение от начала отсчета для печати перекрытия.

Состояние между печатью копий

ИД ATTR_BTWNCPYSTS

Тип String

Описание
Указывает, находится ли загрузчик в состоянии, когда печать предыдущей копии буферного файла уже завершена, а следующей - еще не начата. Возвращаемые значения: *YES, *NO.

Состояние между печатью файлов

ИД ATTR_BTWNFILESTS

Тип String

Описание
Указывает, находится ли загрузчик в состоянии, когда обработка предыдущего файла уже завершена, а следующего - еще не начата. Возвращаемые значения: *YES, *NO.

Изменения

ИД ATTR_CHANGES

Тип String

Описание
Время вступления в силу ожидающих изменений. Возможные значения: *NORDYF, *FILEEND или пробел, означающий, что изменения вступают в силу немедленно.

Число символов на дюйм

ИД ATTR_CPI

Тип Float

Описание
Число символов на дюйм по горизонтали.

Кодовая страница

ИД ATTR_CODEPAGE

Тип String

Описание
Задает таблицу преобразования графических символов в кодовые знаки для данного буферного файла. Если в качестве набора графических символов указано специальное значение, то этот атрибут может равняться нулю.

Имя кодированного шрифта

ИД ATTR_CODEDFNT

Тип String

Описание
Имя кодированного шрифта. Кодированный шрифт - это ресурс AFP, состоящий из набора символов и кодовой страницы. Специальные значения: *FNTCHRSET.

Библиотека кодированного шрифта

ИД ATTR_CODEDFNTLIB

Тип String

Описание
Имя библиотеки, содержащей кодированный шрифт. Если в качестве кодированного шрифта задано специальное значение, то это поле может содержать пустое значение.

Управляющий символ

ИД ATTR_CONTROLCHAR

Тип String

Описание
Указывает, применяется ли в данном файле управляющий символ принтера Американского национального института стандартов. Возможные значения: *NONE, если данные для печати не содержат управляющих символов, или *FCFC, если каждая запись начинается с управляющего символа.

Преобразование строковых данных

ИД ATTR_CONVERT_LINEDATA

Тип String

Описание
Указывает, преобразуются ли строковые данные в формат AFPDS перед записью в буфер. Возможные значения: *YES и *NO.

Число копий

ИД ATTR_COPIES

Тип Integer

Описание
Общее число напечатанных копий буферного файла.

Осталось напечатать копий

ИД ATTR_COPIESLEFT

Тип Integer

Описание
Число копий буферного файла, которые осталось напечатать.

Угол скрепления

ИД ATTR_CORNER_STAPLE

Тип String

Описание
Базовый угол для скрепления. Скрепка помещается в базовый угол листа. Возможные значения: *NONE, *DEVD, *BOTRIGHT, *TOPRIGHT, *TOPLEFT и *BOTLEFT.

Текущая страница

ИД ATTR_CURPAGE

Тип Integer

Описание
Страница, которая обрабатывается в данный момент заданием загрузчика.

Формат данных

ИД ATTR_DATAFORMAT

Тип String

Описание
Формат данных. Возможные значения: *RCDDATA и *ALLDATA.

Очередь данных

ИД ATTR_DATA_QUEUE

Тип String

Описание
Задает имя очереди данных, связанной с очередью вывода, в интегрированной файловой системе, либо значение *NONE″, если с очередью вывода не связана очередь данных. Имя очереди данных задается в формате ″/QSYS.LIB/библиотека.LIB/очередь-данных.DTAQ″, где *библиотека* - имя библиотеки, в которой расположена очередь данных, а *очередь-данных* - имя очереди данных.

Время открытия файла данных

ИД ATTR_DATE

Тип String

Описание

Для буферных файлов - дата, когда был открыт буферный файл. Для ресурсов AFP - дата последнего изменения объекта. Дата представляет собой символьную строку следующего вида: В ГГ ММ ДД.

Конечная дата создания задания буферного файла

ИД ATTR_DATE_END

Тип String

Описание

Конечная дата завершения задания системы, в котором создавался буферный файл. Если в поле Начальная дата создания задания буферного файла указано значение *ALL, данное поле должно быть пустым. Если в поле Начальная дата создания задания буферного файла указана дата, в данном поле также должна быть указана допустимая дата. Формат даты: CYYMMDD. Возможные значения:

- *LAST: Будут возвращены все буферные файлы, созданные после указанной начальной даты.
- Дата: Будут возвращены все буферные файлы, созданные между указанными начальной и конечной датами.

Расшифровка формата даты CYYMMDD:

- C - столетие, 0 обозначает годы 19xx, 1 обозначает годы 20xx
- YY - год
- MM - месяц
- DD - день

Дата начала обработки буферного файла загрузчиком

ИД ATTR_DATE_WTR_BEGAN_FILE

Тип String

Описание

Указывает дату, когда загрузчик начал обработку данного буферного файла. Дата представляет собой символьную строку следующего формата: В ГГ ММ ДД.

Дата завершения обработки буферного файла загрузчиком

ИД ATTR_DATE_WTR_CMPL_FILE

Тип String

Описание

Указывает дату, когда загрузчик закончил обработку данного буферного файла. Дата представляет собой символьную строку следующего формата: В ГГ ММ ДД.

Пользовательские данные DBCS

ИД ATTR_DBCSDATA

Тип String

Описание

Указывает, содержит ли буферный файл данные DBCS. Возможные значения: *YES и *NO.

Символы расширенного DBCS

ИД ATTR_DBCSEXTENSN

Тип String

Описание

Указывает, поддерживает ли система символы расширенного DBCS. Возможные значения: *YES и *NO.

Поворот символов DBCS

ИД ATTR_DBCAROTATE

Тип String

Описание

Указывает, будут ли символы DBCS при печати поворачиваться на 90 градусов против часовой стрелки. Возможные значения: *YES и *NO.

Число символов DBCS на дюйм

ИД ATTR_DBCSCPI

Тип Integer

Описание

Число двухбайтовых символов на дюйм. Возможные значения: -1, -2, 5, 6 и 10. -1 соответствует значению *CPI. -2 соответствует значению *CONDENSED.

Пробелы вместо скобочных символов DBCS

ИД ATTR_DBCSSISO

Тип String

Описание

Задает представление открывающих и закрывающих символов при печати. Возможные значения: *NO, *YES и *RIGHT.

Запись с отсрочкой

ИД ATTR_DFR_WRITE

Тип String

Описание

Указывает, будут ли данные для печати блокированы в буферах системы до

Угол поворота страницы

ИД ATTR_PAGRTT

Тип Integer

Описание

Угол поворота текста на странице относительно того, как форма загружена в принтер. Возможные значения: -1, -2, -3, 0, 90, 180, 270. -1 соответствует значению *AUTO, -2 соответствует значению *DEVD, -3 соответствует значению *COR.

Удаление файла после отправки

ИД ATTR_DELETESPLF

Тип String

Описание

Указывает, должен ли удаляться буферный файл после отправки на принтер. Возможные значения: *YES и *NO.

Целевая опция

ИД ATTR_DESTOPTION

Тип String

Описание

Целевая опция. Текстовая строка, в которой пользователь может передавать опции в целевую систему.

Целевой тип

ИД ATTR_DESTINATION

Тип String

Описание

Целевой тип. Возможные значения: *OTHER, *AS400, *PSF2.

Класс устройства

ИД ATTR_DEVCLASS

Тип String

Описание

Класс устройства.

Модель устройства

ИД ATTR_DEVMODEL

Тип String

Описание

Номер модели устройства.

Состояние устройства

ИД ATTR_DEVSTATUS

Тип Integer

Описание

Состояние принтера. Возможные значения: 0 (выключен), 10 (выключается), 20 (включается), 30 (включен), 40 (устанавливается соединение), 60 (работает), 66 (работает загрузчик), 70 (блокирован), 75 (питание отключено), 80 (восстанавливается), 90 (восстановление отменено), 100 (сбой), 106 (сбой загрузчика), 110 (обслуживается), 111 (неисправен), 112 (занят), 113 (неизвестно).

Тип устройства

ИД ATTR_DEVTYPE

Тип String

Описание

Тип устройства.

Просмотр всех файлов

ИД ATTR_DISPLAYANY

Тип String

Описание
Указывает, могут ли пользователи с правами на чтение данных из очереди вывода просматривать содержимое всех файлов вывода или только тех из них, которые им принадлежат. Возможные значения: *YES, *NO и *OWNER.

Лоток для разделительных страниц

ИД ATTR_DRWRSEP

Тип Integer

Описание
Задает лоток для подачи разделительных страниц между обработкой различных заданий и файлов. Возможные значения: -1, -2, 1, 2, 3. -1 соответствует значению *FILE; -2 - значению *DEVD.

Число скрепок бокового скрепления

ИД ATTR_EDGESTITCH_NUMSTAPLES

Тип Integer

Описание
Число скрепок, которые вставляются вдоль оси скрепления.

Базовый край бокового скрепления

ИД ATTR_EDGESTITCH_REF

Тип String

Описание
Определяет расположение скрепок на странице, вставляемых вдоль оси скрепления. Возможные значения: *NONE, *DEVD, *BOTTOM, *RIGHT, *TOP и *LEFT.

Смещение от базового края бокового скрепления

ИД ATTR_EDGESTITCH_REFOFF

Тип Float

Описание
Смещение линии скрепления от базового края бокового скрепления по направлению к центру листа.

Состояние Завершен (ожидание)

ИД ATTR_ENDPNDSTS

Тип String

Описание
Указывает, была ли вызвана команда Завершить работу загрузчика (ENDWTR) для данного загрузчика. Возможные значения: *NO - команда ENDWTR не вызывалась, *IMMED - работа загрузчика завершится, когда не останется данных в буферах вывода, *CTRLD - работа загрузчика будет завершена по окончании печати данной копии буферного файла, *PAGEEND - работа загрузчика будет завершена по окончании печати страницы.

Последняя страница

ИД ATTR_ENDPAGE

Тип Integer

Описание

Номер страницы буферного файла, которая должна быть напечатана последней. Возможные значения: 0 или номер последней страницы. 0 соответствует значению *END.

Размер конвертов

ИД ATTR_ENVLP_SOURCE

Тип String

Описание

Размер конверта в источнике конвертов. Если значение в этом поле не указано или указано недопустимое значение, применяется специальное значение *MFRTYPMDL. Возможные значения: *NONE - источник конвертов не применяется, *MFRTYPMDL - размер конверта определяется типом и моделью принтера, *MONARCH (3.875 x 7.5 дюймов), *NUMBER9 (3.875 x 8.875 дюймов), *NUMBER10 (4.125 x 9.5 дюймов), *B5 (176 мм x 250 мм), *C5 (162 мм x 229 мм), *DL (11 мм x 220 мм).

Число разделительных страниц для файлов

ИД ATTR_FILESEP

Тип Integer

Описание

Число разделительных страниц, помещаемых в начале каждой копии буферного файла. Возможные значения: -1 или число разделительных страниц. -1 соответствует значению *FILE.

Перенос записей

ИД ATTR_FOLDREC

Тип String

Описание

Переносятся ли строки, длина которых превышает ширину страницы, на следующую строку. Возможные значения: *YES и *NO.

Идентификатор шрифта

ИД ATTR_FONTID

Тип String

Описание

Применяемый шрифт принтера. Возможные специальные значения: *CPI и *DEVD.

Определение формы

ИД ATTR_FORM_DEFINITION

Тип String

Описание

Имя определения формы в интегрированной файловой системе или специальное значение. Имя задается в формате ″/QSYS.LIB/библиотека.LIB/определение-формы.FORMDF″, где *библиотека* - имя библиотеки, в которой расположено определение формы, а *определение-формы* - имя определения формы. Возможные специальные значения: *NONE, *INLINE, *INLINED и *DEVD.

Подача бумаги

ИД ATTR_FORMFEED

Тип String

Описание
Способ подачи бумаги на принтер. Возможные значения: *CONT, *CUT, *AUTOCUT и *DEVD.

Тип формы

ИД ATTR_FORMTYPE

Тип String

Описание
Тип формы, который будет загружен на принтер для печати буферного файла.

Опция отправки сообщений о форме

ИД ATTR_FORMTYPEMSG

Тип String

Описание
Опция отправки сообщения в очередь сообщений загрузчика при завершении обработки текущей формы. Возможные значения: *MSG, *NOMSG, *INFOMSG и *INQMSG.

Горизонтальный отступ на лицевой стороне

ИД ATTR_FTMGN_ACR

Тип Float

Описание
Задает отступ от левого края при печати на лицевой стороне листа. -2 соответствует специальному значению *DEVD.

Вертикальный отступ на лицевой стороне

ИД ATTR_FTMGN_DWN

Тип Float

Описание
Задает отступ от верхнего края при печати на лицевой стороне листа. -2 соответствует специальному значению *DEVD.

Перекрытие на лицевой стороне

ИД ATTR_FRONT_OVERLAY

Тип String

Описание
Имя перекрытия на лицевой стороне в интегрированной файловой системе. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/перекрытие.OVL″, где *библиотека* - имя библиотеки, в которой расположен ресурс, а *перекрытие* - имя перекрытия. Значение *NONE указывает, что перекрытие на лицевой стороне не задано.

Горизонтальное смещение перекрытия на лицевой стороне

ИД ATTR_FTOVL_ACR

Тип Float

Описание
Горизонтальное смещение от начала отсчета для печати перекрытия.

Вертикальное смещение перекрытия на лицевой стороне

ИД ATTR_FTOVL_DWN

Тип Float

Описание
Вертикальное смещение от начала отсчета для печати перекрытия.

Набор графических символов

ИД ATTR_CHAR_ID

Тип String

Описание
Набор графических символов, который будет применяться при печати данного файла. Возможные специальные значения: *DEVD, *SYSVAL и *JOBCCSID.

Аппаратное выравнивание

ИД ATTR_JUSTIFY

Тип Integer

Описание
Процентное соотношение, в котором данные вывода выровнены по правому краю. Возможные значения: 0, 50 и 100.

Состояние Блокирован

ИД ATTR_HELDSTS

Тип String

Описание
Указывает, блокирован ли загрузчик. Возможные значения: *YES и *NO.

Блокировка буферного файла

ИД ATTR_HOLD

Тип String

Описание
Указывает, блокирован ли буферный файл. Возможные значения: *YES и *NO.

Состояние Блокирован (ожидание)

ИД ATTR_HOLDPNDSTS

Тип String

Описание
Указывает, была ли вызвана команда Блокировать загрузчик (HLDWTR) для данного загрузчика. Возможные значения: *NO - команда HLDWTR не вызывалась, *IMMED - загрузчик будет блокирован, когда в его буферах вывода не останется данных, *CTRLD - загрузчик будет блокирован по окончании печати текущей копии буферного файла, *PAGEEND - загрузчик будет блокирован после обработки страницы.

Конфигурация изображения

ИД ATTR_IMGCFG

Тип String

Описание
Параметры преобразования для различных форматов изображений и потоков данных.

Инициализация загрузчика

ИД ATTR_WTRINIT

Тип String

Описание
Пользователь может задать условия инициализации принтера. Возможные значения: *WTR, *FIRST и *ALL.

IP-адрес

ИД ATTR_INTERNETADDR

Тип String

Описание
IP-адрес целевой системы.

Набор символов атрибутов IPP

ИД ATTR_IPP_ATTR_CHARSET

Тип String

Описание
Задает набор символов (кодированный набор символов и кодировку), связанный с атрибутами буферного файла, заданными IPP.

ИД задания IPP

ИД ATTR_IPP_JOB_ID

Тип Integer

Описание
ИД задания IPP, связанного с принтером IPP, создавшем задание.

Имя задания IPP

ИД ATTR_IPP_ATR_CHARSET

Тип String

Описание
Интуитивно понятное имя задания.

Язык имени задания IPP

ИД ATTR_IPP_JOB_NAME_NL

Тип String

Описание
Идентификатор языка, связанный с именем задания.

Имя пользователя, запустившего задание IPP

ИД ATTR_IPP_JOB_ORIGUSER

Тип String

Описание
Задает пользователя, запустившего данное задание IPP.

Язык имени пользователя, запустившего задание IPP

ИД ATTR_IPP_JOB_ORIGUSER_NL

Тип String

Описание
Задает идентификатор языка, связанный с именем пользователя, запустившего задание.

Имя принтера IPP

ИД ATTR_IPP_PRINTER_NAME

Тип String

Описание
Задает принтер IPP, создавший это задание.

Имя задания

ИД ATTR_JOBNAME

Тип String

Описание
Имя задания, создавшего буферный файл.

Номер задания

ИД ATTR_JOBNUMBER

Тип String

Описание
Номер задания, создавшего буферный файл.

Число разделителей заданий

ИД ATTR_JOBSEPRATR

Тип Integer

Описание
Число разделителей заданий, которые будут помещаться в начало вывода задания, буферные файлы которого расположены в данной очереди вывода. Возможные значения: -2 и 0-9. -2 соответствует значению *MSG. Разделитель заданий задается при создании очереди вывода.

Задание системы

ИД ATTR_JOBSYSTEM

Тип String

Описание
Задание системы, в котором был создан буферный файл.

Пользователь задания

ИД ATTR_JOBUSER

Тип String

Описание
Имя пользователя, создавшего буферный файл.

Последняя напечатанная страница

ИД ATTR_LASTPAGE

Тип Integer

Описание
Номер последней напечатанной страницы в файле, если был напечатан не весь вывод задания.

Длина страницы

ИД ATTR_PAGELEN

Тип Float

Описание
Длина страницы. Единица измерения задается в атрибуте ″способ измерения″.

Имя библиотеки

ИД ATTR_LIBRARY

Тип String

Описание
Имя библиотеки.

Число строк на дюйм

ИД ATTR_LPI

Тип Float

Описание
Число строк на дюйм по вертикали в буферном файле.

Межстрочный интервал

ИД ATTR_LINESPACING

Тип String

Описание
Межстрочный интервал при печати данных файла. Этот атрибут устанавливается только для файлов принтеров *LINE и *AFPDSLINE. Возможные значения: *SINGLE, *DOUBLE, *TRIPLE и *CTLCHAR.

Производитель, тип и модель

ИД ATTR_MFGTYPE

Тип String

Описание
Задает производителя, тип и модель. Эти значения применяются при преобразовании печатаемых данных из формата SCS в ASCII.

Максимальное число заданий в списке клиента

ИД ATTR_MAX_JOBS_PER_CLIENT

Тип Integer

Описание
Ограничение на размер очереди принтера, задаваемое клиентом.

Максимальное число записей буферизованного вывода

ИД ATTR_MAXRECORDS

Тип Integer

Описание
Ограничение на число записей в файле, которое применялось на момент открытия файла. 0 соответствует значению *NOMAX.

Способ измерения

ИД ATTR_MEASMETHOD

Тип String

Описание
Способ измерения, применяемый для атрибутов длины и ширины страницы. Возможные значения: *ROWCOL и *UOM.

Справка по сообщению

ИД ATTR_MSGHELP

Тип char(*)

Описание
Справку по сообщению, также называемую текстом второго уровня, можно получить, отправив запрос на получение сообщения. Для размера справки по сообщению установлено системное ограничение в 3000 символов (для английской версии - на 30 % меньше с запасом для перевода).

ИД сообщения

ИД ATTR_MESSAGEID

Тип String

Описание
ИД сообщения.

Очередь сообщений

ИД ATTR_MESSAGE_QUEUE

Тип String

Описание
Имя очереди сообщений, которая применяется загрузчиком для отправки сообщений о выполнении, в интегрированной файловой системе. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/очередь-сообщений.MSGQ″, где *библиотека* - библиотека, содержащая очередь сообщений, а *очередь-сообщений* - имя очереди сообщений.

Ответ на сообщение

ИД ATTR_MSGREPLY

Тип String

Описание
Ответ на сообщение. Строка, которая отправляется клиентом в качестве ответа на сообщение-вопрос. При получении сообщения сервер возвращает значение атрибута, представляющее собой ответ по умолчанию, которым может воспользоваться клиент. Для его

размера установлено системное ограничение в 132 символа. Поскольку сообщение имеет изменяемую длину, оно должно оканчиваться символом NULL.

Текст сообщения

ИД ATTR_MSGTEXT

Тип String

Описание

Текст сообщения, называемый также текстом первого уровня, можно получить с помощью запроса на получение сообщения. Для его размера установлено системное ограничение в 132 символа.

Тип сообщения

ИД ATTR_MSGTYPE

Тип String

Описание

Тип сообщения, представляющий собой двузначное число в кодировке EBCDIC. Все сообщения разделены на два типа в зависимости от того, требуется ли ответ на сообщение: ’04’ - информационные сообщения, не требующие ответа (вместо отправки ответа может потребоваться корректирующее действие), ’05’ - сообщения-вопросы, на которые получатель должен отправить ответ.

Серьезность сообщения

ИД ATTR_MSGSEV

Тип Integer

Описание

Серьезность сообщения. Допустимы значения от 00 до 99. Чем больше это значение, тем важнее связанное с ним сообщение.

Поддержка составных ответов

ИД ATTR_MULTI_ITEM_REPLY

Тип String

Описание

Если клиент присвоит этому атрибуту значение *YES, значительно повысится эффективность выполнения операций над списком буферных файлов. Значение по умолчанию - *NO.

Идентификатор сети

ИД ATTR_NETWORK

Тип String

Описание

Идентификатор сети той системы, в которой был создан файл.

Число байт в буферном файле

ИД ATTR_NUMBYTES_SPLF

Тип Integer

Описание

Размер буферного или потокового файла в байтах. Это значение задает размер файла до

преобразования данных. Для файлов, размер которых превышает 2**31 - 1 байт, в этом атрибуте указывается размер, поделенный на 10 Кб. Этот атрибут не применяется для буферных файлов, просматриваемых в постраничном режиме.

Число байт для чтения/записи

ИД ATTR_NUMBYTES

Тип Integer

Описание
Число байт, которое должно быть прочитано или записано. Значение этого атрибута интерпретируется в зависимости от того, какая операция выполняется над объектом.

Число файлов

ИД ATTR_NUMFILES

Тип Integer

Описание
Число буферных файлов в очереди вывода.

Число загрузчиков, запущенных в очереди

ИД ATTR_NUMWRITERS

Тип Integer

Описание
Число заданий загрузчиков, запущенных в очереди вывода.

Расширенный атрибут объекта

ИД ATTR_OBJEXTATTR

Тип String

Описание
Атрибут extended, который устанавливается для некоторых объектов, например, ресурсов шрифта. Это значение можно просмотреть на сервере с помощью команд WRKOBJ и DSPOBJD. В названии меню сервера может быть указано только слово Атрибут. Для ресурсов шрифта значение атрибута обычно равно CDEPAG, CDEFNT или FNTCHRSET.

Состояние В очереди заданий

ИД ATTR_ONJOBQSTS

Тип String

Описание
Указывает, находится ли загрузчик в очереди заданий (в данный момент не работает). Возможные значения: *YES и *NO.

Начальные команды

ИД ATTR_OPENCMDS

Тип String

Описание
Указывает, нужно ли добавить в поток данных перед содержимым буферного файла начальные команды SCS. Возможные значения: *YES и *NO.

Управляется оператором

ИД ATTR_OPCNTRL

Тип String

Описание
Указывает, могут ли пользователи с правами на управление заданием работать с буферными файлами в данной очереди. Возможные значения: *YES и *NO.

Порядок файлов в очереди

ИД ATTR_ORDER

Тип String

Описание
Задает порядок буферных файлов в очереди вывода. Возможные значения: *FIFO и *JOBNBR.

Принимающий лоток

ИД ATTR_OUTPUTBIN

Тип Integer

Описание
Принимающий лоток принтера. Допустимы значения от 1 до 65535. 0 соответствует специальному значению *DEVD.

Приоритет вывода

ИД ATTR_OUTPTY

Тип String

Описание
Приоритет буферного файла. Допустимы значения от 1 (максимальный приоритет) до 9 (минимальный приоритет). Возможные значения: 0-9, где 0 представляет значение *JOB.

Очередь вывода

ИД ATTR_OUTPUT_QUEUE

Тип String

Описание
Имя очереди вывода в интегрированной файловой системе. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/очередь.OUTQ″, где *библиотека* - библиотека, в которой расположена очередь вывода, а *очередь* - имя очереди вывода.

Состояние очереди вывода

ИД ATTR_OUTQSTS

Тип String

Описание
Состояние очереди вывода. Возможные значения: RELEASED и HELD.

Общее состояние

ИД ATTR_OVERALLSTS

Тип Integer

Описание

Общее состояние ″логического принтера″. ″Логический принтер″ представляет принтер, очередь вывода и задание загрузчика. Возможные значения: 1 (недоступен), 2 (выключен или еще недоступен), 3 (остановлен), 4 (ожидает ответ на сообщение), 5 (блокирован), 6 (завершается), 7 (блокирован - ожидание), 8 (ожидание принтера), 9 (ожидание запуска), 10 (печать), 11 (ожидание очереди вывода), 12 (устанавливается соединение), 13 (питание отключено), 14 (неисправен), 15 (обслуживается), 999 (неизвестно).

Номер строки переполнения

ИД ATTR_OVERFLOW

Тип Integer

Описание

Последняя строка, которая будет напечатана перед переносом данных на следующую страницу.

Постраничный просмотр

ИД ATTR_PAGE_AT_A_TIME

Тип String

Описание

Определяет, будет ли буферный файл открыт для просмотра в постраничном режиме. Возможные значения: *YES и *NO.

Примерное число страниц

ИД ATTR_PAGES_EST

Тип String

Описание

Указывает, является ли число страниц приблизительным или точным. Возможные значения: *YES и *NO.

Определение страницы

ИД ATTR_PAGE_DEFINITION

Тип String

Описание

Имя определения страницы в интегрированной файловой системе или специальное значение. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/определение-страницы.PAGDFN″, где *библиотека* - это имя библиотеки, в которой расположено определение страницы, а *определение-страницы* - это имя определения страницы. Возможные специальные значения: *NONE.

Номер страницы

ИД ATTR_PAGENUMBER

Тип Integer

Описание

Номер страницы, которую нужно считать из буферного файла, открытого в режиме постраничного вывода.

Число страниц на стороне

ИД ATTR_MULTIUP

Тип Integer

Описание

Число логических страниц файла, которое будет печататься на каждой стороне листа бумаги. Возможные значения: 1, 2 и 4.

Источник бумаги 1

ИД ATTR_PAPER_SOURCE_1

Тип String

Описание

Размер бумаги в источнике бумаги 1. Если значение в этом поле не указано или указано недопустимое значение, применяется специальное значение *MFRTYPMDL. Возможные значения: *NONE - в источнике бумаги 1 нет бумаги, или бумага подается на принтер вручную, *MFRTYPMDL - размер бумаги определяется производителем, моделью и типом принтера, *LETTER (8.5 x 11.0 дюймов), *LEGAL (8.5 x 14.0 дюйма), *EXECUTIVE (7.25 x 10.5 дюйма), *LEDGER (17.0 x 11.0 дюйма), *A3 (297 мм x 420 мм), *A4 (210 мм x 297 мм), *A5 (148 мм x 210 мм), *B4 (257 мм x 364 мм), *B5 (182 мм x 257 мм), *CONT80 (бумажная лента шириной 8.0 дюймов), *CONT132 (бумажная лента шириной 13.2 дюйма).

Источник бумаги 2

ИД ATTR_PAPER_SOURCE_2

Тип String

Описание

Размер бумаги в источнике бумаги 2. Если значение в этом поле не указано или указано недопустимое значение, применяется специальное значение *MFRTYPMDL. Возможные значения: *NONE - в источнике бумаги 2 нет бумаги, или бумага подается на принтер вручную, *MFRTYPMDL - размер бумаги определяется производителем, моделью и типом принтера, *LETTER (8.5 x 11.0 дюйма), *LEGAL (8.5 x 14.0 дюйма), *EXECUTIVE (7.25 x 10.5 дюйма), *LEDGER (17.0 x 11.0 дюйма), *A3 (297 мм x 420 мм), *A4 (210 мм x 297 мм), *A5 (148 мм x 210 мм), *B4 (257 мм x 364 мм), *B5 (182 мм x 257 мм), *CONT80 (бумажная лента шириной 8.0 дюйма), *CONT132 (бумажная лента шириной 13.2 дюйма).

Размер в пикселах

ИД ATTR_PELDENSITY

Тип String

Описание

Этот атрибут устанавливается только для ресурсов шрифта. Он задает размер в пикселах (1 - 240 пикселов, а 2 - 320 пикселов). В будущем на сервере могут быть определены и другие значения этого параметра.

Размер в пунктах

ИД ATTR_POINTSIZE

Тип Float

Описание

Размер в пунктах для печати содержимого данного буферного файла. 0 соответствует специальному значению *NONE.

Точность печати

ИД ATTR_FIDELITY

Тип String

Описание
Способ обработки ошибок при печати. Возможные значения: *ABSOLUTE и *CONTENT.

Печать на обеих сторонах

ИД ATTR_DUPLEX

Тип String

Описание
Способ печати данных. Возможные значения: *FORMDF, *NO, *YES и *TUMBLE.

Качество печати

ИД ATTR_PRTQUALITY

Тип String

Описание
Качество печати данного буферного файла. Возможные значения: *STD, *DRAFT, *NLQ и *FASTDRAFT.

Порядок вывода на печать

ИД ATTR_PRTSEQUENCE

Тип String

Описание
Порядок вывода данных на печать. Возможные значения: *NEXT.

Текст для печати

ИД ATTR_PRTTEXT

Тип String

Описание
Текст, который печатается внизу каждой страницы и на разделительных страницах. Возможные специальные значения: *BLANK и *JOB.

Принтер

ИД ATTR_PRINTER

Тип String

Описание
Имя принтера.

Назначенный принтер

ИД ATTR_PRTASSIGNED

Тип String

Описание
Указывает, назначен ли принтер. Возможные значения: 1 (назначен определенному принтеру), 2 (назначен нескольким принтерам), 3 (не назначен).

Тип принтера

ИД ATTR_PRTDEVTYPE

Тип String

Описание
Тип потока данных принтера. Возможные значения: *SCS, *IPDS, *USERASCII, *AFPDS и *LINE.

Файл принтера

ИД ATTR_PRINTER_FILE

Тип String

Описание
Имя файла принтера в интегрированной файловой системе. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/файл-принтера.FILE″, где *библиотека* - библиотека, в которой расположен файл принтера, а *файл-принтера* - имя файла принтера.

Очередь принтера

ИД ATTR_RMTPRTQ

Тип String

Описание
Имя очереди целевого принтера, применяемое при отправке буферных файлов с помощью SNDTCPSPLF (LPR).

Информация о публикации - Поддержка цветной печати

ИД ATTR_PUBINF_COLOR_SUP

Тип String

Описание
Указывает в информации о публикации поддержку цвета.

Информация о публикации - Число страниц в минуту (цветная печать)

ИД ATTR_PUBINF_PPM_COLOR

Тип Integer

Описание
Указывает в информации о публикации число страниц в минуту, поддерживаемое при цветной печати.

Информация о публикации - Число страниц в минуту (одноцветная печать)

ИД ATTR_PUBINF_PPM

Тип Integer

Описание
Указывает в информации о публикации число страниц в минуту, поддерживаемое при монохромной печати.

Информация о публикации - Поддержка двусторонней печати

ИД ATTR_PUBINF_DUPLEX_SUP

Тип String

Описание
Указывает в информации о публикации поддержку двусторонней печати.

Информация о публикации - Расположение

ИД ATTR_PUBINF_LOCATION

Тип String

Описание
Указывает в информации о публикации расположение описания.

Имя удаленного расположения

ИД ATTR_RMTLOCNAME

Тип String

Описание
Имя расположения принтера.

Длина записи

ИД ATTR_RECLENGTH

Тип Integer

Описание
Длина записи.

Сокращать вывод

ИД ATTR_REDUCE

Тип String

Описание
Способ печати нескольких логических страниц на каждой стороне физического листа. Возможные значения: *TEXT и ????.

Удаленная система

ИД ATTR_RMTSYSTEM

Тип String

Описание
Имя удаленной системы. Возможные специальные значения: *INTNETADR.

Заменять непечатаемые символы

ИД ATTR_RPLUNPRT

Тип String

Описание
Указывает, будут ли заменяться непечатаемые символы. Возможные значения: *YES и *NO.

Символ замещения

ИД ATTR_RPLCHAR

Тип String

Описание
Символ, замещающий непечатаемые символы.

Повторить печать

ИД ATTR_RESTART

Тип Integer

Описание
Повторить печать. Возможные значения: -1, -2, -3 или номер страницы, начиная с которой нужно повторить печать. -1 соответствует значению *STRPAGE, -2 - значению *ENDPAGE, а -3 - значению *NEXT.

Число скрепок скрепления посередине

ИД ATTR_SADDLESTITCH_NUMSTAPLES

Тип Integer

Описание
Число скрепок, которые вставляются вдоль оси скрепления.

Базовый край скрепления посередине

ИД ATTR_SADDLESTITCH_REF

Тип String

Описание
Одна или несколько скрепок вставляются в лист вдоль оси скрепления, расположенной по центру листа параллельно базовому краю. Возможные значения: *NONE, *DEVD, *TOP и *LEFT.

Сохранить буферный файл

ИД ATTR_SAVESPLF

Тип String

Описание
Указывает, следует ли сохранять буферный файл после загрузки. Возможные значения: *YES и *NO.

Начало отсчета

ИД ATTR_SEEKOFF

Тип Integer

Описание
Смещение указателя. Допустимы как положительные, так и отрицательные значения. Начало отсчета задается в соответствующем атрибуте.

Текущее положение указателя

ИД ATTR_SEEKORG

Тип Integer

Описание
Допустимы значения 1 (начало документа), 2 (текущая позиция), 3 (конец документа).

Приоритет отправки

ИД ATTR_SENDPTY

Тип String

Описание
Приоритет отправки. Возможные значения: *NORMAL и *HIGH.

Разделительная страница

ИД ATTR_SEPPAGE

Тип String

Описание
Указывает, нужно ли печатать титульную страницу. Возможные значения: *YES и *NO.

Исходный лоток

ИД ATTR_SRCDRWR

Тип Integer

Описание
Лоток, который будет применяться, если выбрана опция автоматической подачи отдельных листов бумаги. Возможные значения: -1, -2, 1-255. -1 соответствует значению *E1, -2 - значению *FORMDF.

SCS буфера

ИД ATTR_SPLSCS

Тип Long

Описание
Определяет способ применения данных SCS при создании буферного файла.

Поместить данные в буфер

ИД ATTR_SPOOL

Тип String

Описание
Указывает, помещается ли печатаемый вывод в буфер. Возможные значения: *YES и *NO.

Способ идентификации при создании буферного файла

ИД ATTR_SPLF_AUTH_METHOD

Тип Integer

Описание
Задает способ идентификации клиента, применяемого для создания буферного файла. Возможные значения: x'00'(*NONE), x'01'(*REQUESTER), x'02'(*BASIC), x'03'(*CERTIFICATE) и 'x'04'(*DIGEST).

Способ защиты для создания буферного файла

ИД ATTR_SPLF_SECURITY_METHOD

Тип String

Описание
Задает способ защиты, применяемый при создании буферного файла. Возможные значения: x'00'(*NONE), x'01'(*SSL3) и x'02'(*TLS).

Имя буферного файла

ИД ATTR_SPOOLFILE

Тип String

Описание
Имя буферного файла.

Номер буферного файла

ИД ATTR_SPLFNUM

Тип Integer

Описание

Номер буферного файла. Возможные специальные значения: -1 и 0. Значению *LAST соответствует -1, значению *ONLY - 0.

Состояние буферного файла

ИД ATTR_SPLFSTATUS

Тип String

Описание

Состояние буферного файла. Возможные значения: *CLOSED, *HELD, *MESSAGE, *OPEN, *PENDING, *PRINTER, *READY, *SAVED и *WRITING.

Расписание буферизованного вывода

ИД ATTR_SCHEDULE

Тип String

Описание

Этот атрибут устанавливается только для буферных файлов. Он указывает, когда буферный файл станет доступен загрузчику. Возможные значения: *IMMED, *FILEEND и *JOBEND.

Запущен пользователем

ИД ATTR_STARTEDBY

Тип String

Описание

Имя пользователя, запустившего загрузчик.

Первая страница

ИД ATTR_STARTPAGE

Тип Integer

Описание

Номер страницы, с которой начнется печать буферного файла. Возможные значения: -1, 0, 1 или номер страницы. -1 соответствует значению *ENDPAGE. Если указано значение 0, то печать начнется с первой страницы. Если указано значение 1, то будет напечатан весь файл.

Исходная система

ИД ATTR_SYSTEM

Тип String

Описание

Система, в которой был создан буферный файл. Если имя этой системы неизвестно, то будет указано имя целевой системы.

Текстовое описание

ИД ATTR_DESCRIPTION

Тип String

Описание

Описание экземпляра объекта AS400.

Время открытия файла

ИД ATTR_TIMEOPEN

Тип String

Описание

Для буферных файлов - время, когда был открыт буферный файл. Для ресурсов AFP - время последнего изменения объекта. Время представляет собой строку символов следующего формата: ЧЧ ММ СС.

Конечное время создания задания буферного файла

ИД ATTR_TIME_END

Тип String

Описание

Конечное время завершения задания системы, в котором создавался буферный файл. Если в поле Начальная дата создания буферного файла указано *ALL или *LAST, в данном поле должно быть указано пустое значение. Если в поле Конечная дата создания буферного файла указана дата, в данном поле должно быть указано время. Время задается в формате HHMMSS, где:

- HH - Часы
- MM - Минуты
- SS - Секунды

Время начала обработки загрузчиком буферного файла

ИД ATTR_TIME_WTR_BEGAN_FILE

Тип String

Описание

Указывает время, когда загрузчик начал обработку данного буферного файла. Время представляет собой строку символов следующего формата: ЧЧ ММ СС.

Время завершения обработки буферного файла загрузчиком

ИД ATTR_TIME_WTR_CMPL_FILE

Тип String

Описание

Указывает время, когда загрузчик завершил обработку данного буферного файла. Время представляет собой строку символов следующего формата: ЧЧ ММ СС.

Общее число страниц

ИД ATTR_PAGES

Тип Integer

Описание

Число страниц в буферном файле.

Преобразовать SCS в ASCII

ИД ATTR_SCS2ASCII

Тип String

Описание

Указывает, будут ли преобразованы печатаемые данные из формата SCS в ASCII. Возможные значения: *YES и *NO.

Единица измерения

ИД ATTR_UNITOFMEAS

Тип String

Описание
Единица измерения длины. Возможные значения: *CM и *INCH.

Комментарий пользователя

ИД ATTR_USERCMT

Тип String

Описание
Пользовательское описание буферного файла, содержащее не более 100 символов.

Пользовательское описание

ИД ATTR_USERDATA

Тип String

Описание
Краткое описание буферного файла (до 10 символов), заданное пользователем. Возможные специальные значения: *SOURCE.

Пользовательские данные

ИД ATTR_USRDFNDTA

Тип String

Описание
Пользовательские данные, предназначенные для пользовательского приложения или указанной пользователем программы, обрабатывающей буферные файлы. Допустимы любые символы. Максимальный размер - 255.

Пользовательский файл

ИД ATTR_USRDEFFILE

Тип String

Описание
Указывает, был ли буферный файл создан с помощью API. Возможные значения - *YES или *NO.

Пользовательский объект

ИД ATTR_USER_DEFINED_OBJECT

Тип String

Описание
Имя пользовательского объекта, применяемого пользовательским приложением для обработки буферных файлов, в интегрированной файловой системе. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/объект.тип″, где *библиотека* - имя библиотеки, в которой расположен объект, либо специальное значение %LIBL% или %CURLIB%. *объект* - имя объекта, а *тип* - тип объекта. Допустимыми значениями для *типа* являются DTAARA, DTAQ, FILE, PSFCFG, USRIDX, USRQ и USRSPC. Значение *NONE указывает, что пользовательский объект применяться не будет.

Пользовательские опции

ИД ATTR_USEDFNOPTS

Тип String

Описание

Указанные пользователем опции, применяемые пользовательскими приложениями, обрабатывающими буферные файлы. Может быть задано до 4 опций, длина каждого значения - char(10). Допустимы любые символы.

Данные для пользовательского драйвера

ИД ATTR_USRDRVPGMDTA

Тип String

Описание

Указанные пользователем данные, предназначенные для пользовательского драйвера. Допустимы любые символы. Максимальный размер - 5000 символов.

Пользовательский драйвер

ИД ATTR_USER_DRIVER_PROG

Тип String

Описание

Имя пользовательского драйвера, обрабатывающего буферные файлы, в интегрированной файловой системе. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/программа.PGM″, где *библиотека* - имя библиотеки, в которой расположена программа, а *программа* - имя программы. В качестве *библиотеки* можно задать имя библиотеки, либо специальное значение %LIBL% или %CURLIB%. Значение *NONE указывает, что драйвер не задан.

ИД пользователя

ИД ATTR_TOUSERID

Тип String

Описание

ИД пользователя, которому отправлен буферный файл.

Адрес пользователя

ИД ATTR_TOADDRESS

Тип String

Описание

Адрес пользователя, которому отправлен буферный файл.

Пользовательская программа преобразования

ИД ATTR_USER_TRANSFORM_PROG

Тип String

Описание

Имя пользовательской программы преобразования в интегрированной файловой системе. Эта программа выполняет преобразование данных перед их передачей драйверу. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/программа.PGM″, где *библиотека* - имя библиотеки, в которой расположена программа, а *программа* - имя программы. В качестве *библиотеки* можно задать имя библиотеки, либо специальное значение %LIBL% или %CURLIB%. Значение *NONE указывает, что программа преобразования не задана.

Точность просмотра

ИД ATTR_VIEWING_FIDELITY

Тип String

Описание

Определяет процедуру обработки, которая выполняется при просмотре страницы данных буферного файла в постраничном режиме. Возможные значения: *ABSOLUTE и *CONTENT (по умолчанию). Значение *ABSOLUTE означает, что перед выводом страницы будут обработаны нерастровые данные (команды). В случае файлов SCS значение *CONTENT означает, что будут обработаны начальные команды и текущая страница. В случае файлов AFPDS значение *CONTENT означает, что будет обработана первая страница данных и текущая страница.

Класс VM/MVS

ИД ATTR_VMMVSCLASS

Тип String

Описание

Класс VM/MVS. Возможные значения: A-Z и 0-9.

Состояние Ожидание данных

ИД ATTR_WTNGDATASTS

Тип String

Описание

Указывает, ожидает ли загрузчик поступления дополнительных данных после загрузки всего содержимого буферного файла. Возможные значения: *NO - загрузчик не ожидает поступления данных, *YES - загрузчик выполнил обработку всех данных в буферном файле и ожидает поступления новых данных. Загрузчик может перейти в такое состояние при обработке открытого буферного файла с параметром SCHEDULE(*IMMED).

Состояние Ожидание устройства

ИД ATTR_WTNGDEVSTS

Тип String

Описание

Указывает, ожидает ли загрузчик, пока устройство будет освобождено заданием, выполняющим небуферизованный вывод данных на принтер. Возможные значения: *NO - загрузчик не ожидает, когда освободится устройство, *YES - устройство занято, и загрузчик ожидает его освобождения.

Состояние Ожидание сообщения

ИД ATTR_WTNGMSGSTS

Тип String

Описание

Указывает, ожидает ли загрузчик поступления ответа на сообщение-вопрос. Возможные значения: *YES и *NO.

Условие автоматического завершения работы загрузчика

ИД ATTR_WTRAUTOEND

Тип String

Описание

Условия автоматического завершения работы загрузчика (если оно применяется). Возможные значения: *NORDYF и *FILEEND. Значение атрибута Автоматическое завершение работы загрузчика должно быть равно *YES.

Условие завершения работы загрузчика

ИД ATTR_WTREND

Тип String

Описание

Условия завершения работы загрузчика. Возможные значения: *CNTRLD, *IMMED и *PAGEEND. Не следует путать этот атрибут с атрибутом условий автоматического завершения работы загрузчика.

Условие блокирования файла

ИД ATTR_HOLDTYPE

Тип String

Описание

Условия блокирования буферного файла. Возможные значения: *IMMED и *PAGEEND.

Ширина страницы

ИД ATTR_PAGEWIDTH

Тип Float

Описание

Ширина страницы. Единица измерения задается в атрибуте ″способ измерения″.

Объект настройки рабочей станции

ИД ATTR_WORKSTATION_CUST_OBJECT

Тип String

Описание

Имя объекта настройки рабочей станции в интегрированной файловой системе. Это имя задается в формате ″/QSYS.LIB/библиотека.LIB/объект-настройки.WSCST″, где *библиотека* - имя библиотеки, содержащей объект настройки, а *объект-настройки* - имя объекта настройки рабочей станции.

Имя задания загрузчика

ИД ATTR_WRITER

Тип String

Описание

Имя задания загрузчика.

Номер задания загрузчика

ИД ATTR_WTRJOBNUM

Тип String

Описание

Номер задания загрузчика.

Состояние задания загрузчика

ИД ATTR_WTRJOBSTS

Тип String

Описание
Состояние задания загрузчика. Возможные значения: STR, END, JOBQ, HLD и MSGW.

Имя пользователя задания загрузчика

ИД ATTR_WTRJOBUSER

Тип String

Описание
Имя пользователя, запустившего задание загрузчика.

Загрузчик запущен

ИД ATTR_WTRSTRTD

Тип String

Описание
Указывает, запущен ли загрузчик для данного принтера. Возможные значения: 1 - загрузчик запущен, 0 - загрузчик не запущен.

Начальная страница загрузчика

ИД ATTR_WTRSTRPAGE

Тип Integer

Описание
Указывает номер страницы, с которой будет начата печать первого буферного файла при запуске задания загрузчика. Это значение применяется только в том случае, если при запуске загрузчика было указано имя буферного файла.

Состояние Запись данных

ИД ATTR_WRTNGSTS

Тип String

Описание
Указывает, выполняет ли загрузчик принтера запись данных. Возможные значения: *YES - загрузчик выполняет запись данных, *NO - загрузчик находится в другом состоянии, *FILE - загружает разделительные страницы файлов.

Атрибуты объекта сервера сетевой печати

CCSID NPS

ИД ATTR_NPSCCSID

Тип Integer

Описание
CCSID, в котором должны быть закодированы все строковые данные для сервера сетевой печати.

Уровень NPS

ИД ATTR_NPSLEVEL

Описание

Версия, выпуск и уровень модификации сервера сетевой печати. Это значение представляет собой строку символов в формате VXRYMY (например, ″V3R1M0″), где

```
X находится в диапазоне (0..9)
Y находится в диапазоне (0..9, A..Z)
```

Копирование буферных файлов:

You can use the copy method of the SpooledFile class to create a copy of the spooled file that the SpooledFile object represents.

Метод SpooledFile.copy() выполняет следующие операции:

- Создает новый буферный файл в той же очереди вывода и системе, в которых находится исходный буферный файл.
- Возвращает ссылку на созданный буферный файл

SpooledFile.copy() - это новый метод, для применения которого необходимо загрузить JTOpen версии 3.2 или выше, либо применить исправление i5/OS. Рекомендуется загрузить и применять JTOpen. Дополнительная информация приведена в следующем разделе:

- IBM Toolbox for Java и JTOpen: Загрузка файлов 
- IBM Toolbox for Java и JTOpen: Пакеты обновления 

Метод копирования создает точную копию буферного файла, применяя при работе с заданием сетевого сервера печати API Создать буферный файл (QSPCRTSP). Созданную копию буферного файла можно идентифицировать только с помощью уникальных значений даты и времени создания.

Если в качестве параметра метода копирования указать очередь вывода, в начале этой очереди вывода будет создана копия буферного файла. Очередь вывода и исходный буферный файл должны располагаться в одной системе

Пример: Копирование буферного файла с помощью метода SpooledFile.copy()

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Этот пример содержит информацию о создании копии буферного файла в очереди сообщения, содержащей исходный файл, с помощью метода SpooledFile.copy(). Для того чтобы сохранить копию буферного файла в другой очереди сообщения, ее необходимо указать в качестве параметра метода копирования:

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

where <outqname> is the OutputQueue object.

```
public static void main(String args[]) {
   // Создание объекта в системе
   AS400 as400 = new AS400(<systemname>,<username>, <password>);
   // Указание очереди сообщения, содержащей исходный файл.
   OutputQueue outputQueue =
     new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<outqname>.OUTQ");

   // Создание массива, содержащего все элементы, необходимые для
   // uniquely identify a spooled file on the server.
   String[][] splfTags = { {
      <spoolfilename>,
      <spoolfilenum>,
      <jobname>,
      <username>,
      <jobnumber>,
```

```
   // Note that <systemname>,<date>, and <time> are optional.
   // Если они не указаны, необходимо удалить соответствующие теги
   // splfTags[i],[j], где j может принимать значения 5,6 или 7.
   <systemname>,
   <date>,
   <time>},
};

// Сохранение информации, идентифицирующей буферный файл, в файле System.out
for ( int i=0; i<splfTags.length; i++) {
   System.out.println("Copying -> " + splfTags[i][0] + ","
                                    + splfTags[i][1] + ","
                                    + splfTags[i][2] + ","
                                    + splfTags[i][3] + ","
                                    + splfTags[i][4] + ","
                                    + splfTags[i][5] + ","
                                    + splfTags[i][6] + ","
                                    + splfTags[i][7] );

// Создание объекта SpooledFile для исходного буферного файла.
   SpooledFile sourceSpooledFile =
      new SpooledFile(as400,
                      splfTags[i][0],
                      Integer.parseInt(splfTags[i][1]),
                      splfTags[i][2],
                      splfTags[i][3],
                      splfTags[i][5],
                      splfTags[i][6],
                      splfTags[i][7] );
}

// Копирование буферного файла, при котором будет создан объект SpooledFile.
// Для того чтобы сохранить новый буферный файл в конкретной очереди
// сообщений, укажите следующие параметры:
// SpooledFile newSplf = new sourceSpooledFile.copy(<outqname>);
// where <outqname> is an OutputQueue object. Очередь вывода можно
// задать с помощью следующих команд:
// OutputQueue outputQueue =
//    new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<outqname>.OUTQ");
try { SpooledFile newSplf = new sourceSpooledFile.copy();
}

catch (Exception e) {
}
```

SpooledFile Javadoc

Create Spooled File (QSPCRTSP) API

Создание буферных файлов:

You can use the SpooledFileOutputStream class to create new server spooled files. Это класс, производный от стандартного класса java.io.OutputStream JDK; после того, как он будет создан, его можно применять везде, где используется класс OutputStream.

При создании нового класса SpooledFileOutputStream инициатор может указывать следующие параметры:

- Файл принтера, который следует применять
- Очередь вывода, в которую следует помещать буферный файл
- Объект PrintParameterList, который может содержать параметры, переопределяющие значения полей в файле принтера

Все эти параметры необязательны. Если файл принтера не указан, сервер сетевой печати использует файл сетевого принтера по умолчанию (QPNPSPRTF). Параметр очереди вывода указывается здесь ради удобства; его можно задать также в PrintParameterList. Если заданы оба параметра, то значение, указанное в

PrintParameterList, переопределяет значение, указанное в параметре очереди вывода. See the documentation of the SpooledFileOutputStream constructor in the Javadoc for a complete list of which attributes may be set in the PrintParameterList for creating new spooled files.

Use one of the write() methods to write data into the spooled file. Объект SpooledFileOutputStream выполняет буферизацию и отправку данных либо при закрытии потока вывода, либо при заполнении буфера. Буферизация выполняется по двум причинам:

- Она дает возможность анализировать весь буфер целиком и устанавливать тип данных автоматически (см. раздел Типы потоков данных в буферных файлах)
- It makes the output stream work faster because not every write request is communicated to the server

Use the flush() method to force the data to be written to the server.

When the caller is finished writing data to the new spooled file, the close() method is called to close the spooled file. Если буферный файл закрыт, запись в него невозможна. By calling the getSpooledFile() method once the spooled file has been closed, the caller can get a reference to a SpooledFile object that represents the spooled file.

Типы потоков данных в буферных файлах

Для указания типа данных, помещаемых в буферный файл, предназначен атрибут буферного файла Тип данных принтера. Если инициатор не указал тип данных принтера, по умолчанию тип данных устанавливается автоматически. Для этого просматриваются первые несколько тысяч байт буферного файла и определяется, какой архитектуре соответствует поток данных: Поток символов SNA (SCS) или поток данных Advanced Function Printing (AFPDS). После этого устанавливается подходящее значение атрибута. Если данные в буферном файле не соответствуют ни одной из указанных архитектур, для атрибута типа данных устанавливается значение *USERASCII. Автоматическое определение типа данных применимо практически всегда, за исключением некоторых специальных случаев. В этих случаях инициатор может установить для атрибута Тип данных принтера конкретное значение (например, *SCS). Если инициатор использует данные принтера из файла принтера, он должен задать специальное значение *PRTF. При переопределении установленного по умолчанию типа данных во время создания буферного файла необходимо соблюдать осторожность и следить за соответствием типа данных, записываемых в буферный файл, значению атрибута типа данных. Если в файл, предназначенный для приема данных SCS, направляются данные другого типа, система генерирует сообщение об ошибке, а буферный файл теряется.

В общем случае возможны три значения данного атрибута:

- ***SCS** - текстовый поток данных в кодах EBCDIC.
- ***AFPDS** (поток данных Advanced Function Presentation) - еще один поток данных, поддерживаемый на сервере. *AFPDS может содержать текст, изображения и графику, а также использовать внешние ресурсы, такие как перекрытия страниц и внешние изображения на сегментах страницы.
- ***USERASCII** - данные принтера, тип которых не совпадает ни с SCS, ни с AFPDS; сервер просто пересылает такие данные без обработки. Пример данных, направляемых в буферный файл *USERASCII: потоки данных Postscript и HP-PCL.

  SpooledFileOutputStream Javadoc

  “Пример: Создание буферных файлов” на стр. 535
  This example shows how to create a spooled file on a server from an input stream.

  “Пример: Создание буферных файлов SCS” на стр. 536
  В этом примере продемонстрировано применение класса SCS3812Writer для создания потока данных SCS и его записи в буферный файл на сервере.

Генерация потока данных SCS:

Для формирования буферных файлов, которые будут печататься на принтерах, подключенных к серверу, необходимо создать поток данных SCS (поток символов SNA). SCS is a text-based, EBCDIC data stream that

can be printed on SCS printers, IPDS printers, or to PC printers. Перед печатью поток данных SCS может быть преобразован с помощью эмулятора или функции преобразования печати хоста на сервере.

Для генерации потоков данных SCS можно использовать классы загрузчика SCS. Последние преобразуют символы Unicode и опции формата языка Java в поток данных SCS. Существуют пять классов загрузчиков SCS, генерирующих потоки данных SCS разного уровня. Инициатор должен выбрать загрузчик, соответствующий целевому принтеру, на котором он или конечный пользователь будет печатать.

Для генерации потока данных печати SCS используйте следующие классы загрузчиков:

SCS writer class	**Описание**
SCS5256Writer	Простейший класс загрузчика SCS. Поддерживает текстовые символы, символы возврата каретки, смещения строки, новой строки и новой страницы, абсолютное и относительное горизонтальное и вертикальное позиционирование, а также установку вертикального формата.
SCS5224Writer	Расширение загрузчика 5256; поддерживает дополнительные опции установки числа символов на дюйм (CPI) и числа строк на дюйм (LPI).
SCS5219Writer	Расширение загрузчика 5224; поддерживает следующие дополнительные опции: установка левого поля, подчеркивание, типы форм (листы бумаги или конверты), размер формы, качество печати, кодовая страница, набор символов, номер исходного лотка, номер целевого лотка.
SCS5553Writer	Расширение загрузчика 5219; поддерживает следующие дополнительные опции: поворот символа, рисование линий сетки, масштабирование шрифтов. Загрузчик 5553 формирует поток двухбайтовых символов (DBCS).
SCS3812Writer	Расширение загрузчика 5219; поддерживает следующие дополнительные опции: полужирный шрифт, двусторонняя печать, ориентация текста, различные шрифты.

Для конструирования загрузчика SCS инициатору необходимы два параметра: поток вывода и (необязательно) кодировка. Поток данных поступает в поток вывода. При создании буферного файла SCS инициатор сначала конструирует SpooledFileOutputStream (поток вывода буферного файла), а затем на его основе создает объект загрузчика SCS. Параметр кодировки содержит идентификатор целевого кодового набора символов EBCDIC, в который преобразуются символы.

Once the writer is constructed, use the write() methods to output text. Use the carriageReturn(), lineFeed(), and newLine() methods to position the write cursor on the page. Use the endPage() method to end the current page and start a new page.

When all of the data has been written, use the close() method to end the data stream and close the output stream.

Пример

“Пример: Создание буферных файлов SCS” на стр. 536 shows how to generate a SCS data stream using the SCS3812Writer class, and how to write the stream to a spooled file on the server:

Чтение буферных файлов и ресурсов AFP:

You can use the PrintObjectInputStream class to read the raw contents of a spooled file or Advanced Function Printing (AFP) resource from the server. Этот класс представляет собой расширение стандартного класса java.io.InputStream JDK, поэтому он может применяться везде, где применяется класс InputStream.

Obtain a PrintObjectInputStream object by calling either the getInputStream() method on an instance of the SpooledFile class or the getInputStream() method on an instance of the AFPResource class.

Use one of the read() methods for reading from the input stream. Все эти методы возвращают либо число считанных байт, либо -1, если не было считано ни одного байта или был достигнут конец файла.

Use the available() method of PrintObjectInputStream to return the total number of bytes in the spooled file or AFP resource. The PrintObjectInputStream class supports marking the input stream, so PrintObjectInputStream always returns true from the markSupported() method. The caller can use the mark() and reset() methods to move the current read position backward in the input stream. Use the skip() method to move the read position forward in the input stream without reading the data.

Пример

Ниже приведен пример чтения данных из существующего буферного файла на сервере с помощью класса PrintObjectInputStream

Пример: Чтение буферных файлов

PrintObjectInputStream Javadoc

Чтение данных из буферных файлов с помощью PrintObjectPageInputStream и PrintObjectTransformedInputStream:

You can use the PrintObjectPageInputStream class to read the data out of a server AFP and SCS spooled file one page at a time.

You can obtain a PrintObjectPageInputStream object with the getPageInputStream() method.

Use one of the read() methods for reading from the input stream. Все эти методы возвращают либо число считанных байт, либо -1, если не было считано ни одного байта или был обнаружен конец страницы.

Use the available() method of PrintObjectPageInputStream to return the total number of bytes in the current page. The PrintObjectPageInputStream class supports marking the input stream, so PrintObjectPageInputStream always returns true from the markSupported() method. The caller can use the mark() and reset() methods to move the current read position backward in the input stream so that subsequent reads reread the same bytes. The caller can use the skip() method to move the read position forward in the input stream without reading the data.

However, when transforming an entire spooled file data stream is desired, use the PrintObjectTransformedInputStream class.

Пример

Ниже приведен пример преобразований данных, полученных из буферного фала, с помощью классов PrintObjectPageInputStream и PrintObjectTransformedInputStream:

“Пример: Чтение и преобразование буферных файлов” на стр. 539

Информация, связанная с данной

PrintObjectPageInputStream Javadoc

PrintObjectTransformedInputStream Javadoc

Лицензия на продукт

The ProductLicense class enables you to request licenses for products installed on the system. To be compatible with other System i5 license users, the class works through System i5 product license support when requesting or releasing a license.

Класс не применяет стратегию лицензий, но возвращает информацию, достаточную для применения этой стратегии в приложении. При запросе лицензии класс ProductLicense возвращает состояние запроса -- лицензия предоставлена или в лицензии отказано. Если лицензия не предоставлена, приложение должно отключить функцию, для работы с которой необходима эта лицензия, так как в классе IBM Toolbox for Java не хранится информация о том, какую функцию следует отключить.

Use the ProductLicense class with System i license support to enforce the license of your application:

- The server side of your application registers your product and license terms with System i5 license support.
- Приложение-клиент запрашивает и освобождает лицензии с помощью объекта ProductLicense.

Пример: Сценарий работы с классом ProductLicense

Предположим, один из ваших клиентов приобрел 15 неисключительных лицензий на использование продукта. Неисключительные лицензии позволяют одновременно работать с продуктом 15 пользователям, но это не обязательно должны быть одни и те же лица. С продуктом смогут работать любые 15 сотрудников организации. This information is registered with System i license support. При подключении пользователей приложение запрашивает у них лицензии с помощью класса ProductLicense.

- Если с продуктом работает менее 15 человек, лицензия будет предоставлена, и приложение будет запущено.
- При подключении 16-го пользователя запрос ProductLicense не выполняется. При этом приложение выводит сообщение об ошибке и завершает работу.

Когда один из пользователей прекращает работу с приложением, лицензия освобождается с помощью класса ProductLicense. После этого лицензией может воспользоваться другой сотрудник компании.

Информация, связанная с данной

ProductLicense Javadoc

Access package - ProgramCall class

The IBM Toolbox for Java ProgramCall class allows the Java program to call a System i5 program. You can use the ProgramParameter class to specify input, output, and input/output parameters. If the program runs, the output and input/output parameters contain the data that is returned by the System i5 program. If the System i5 program fails to run successfully, the Java program can retrieve any resulting System i5 messages as a list of AS400Message objects.

Обязательные параметры:

- Имя выполняемой программы и параметры
- The AS400 object that represents the server that has the program.

The program name and parameter list can be set on the constructor, through the ProgramCall setProgram() method, or on the run() method The run() method calls the program.

Using the ProgramCall class causes the AS400 object to connect to the server. Информация об управлении соединениями приведена в разделе управление соединениями.

Пример: Применение класса ProgramCall

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведен пример использования класса ProgramCall:

```
                    // Создание объекта AS400.      AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта программы. Программа
// будет запускаться позже.
     ProgramCall pgm = new ProgramCall(sys);
```

```
                        // Задание имени программы.
                        // Так как программа не получает никаких
                        // параметров, в качестве аргумента
                        // ProgramParameter[] передается пустое значение.
     pgm.setProgram(QSYSObjectPathName.toPath("MYLIB", "MYPROG", "PGM"));

                        // Запуск программы.
В этой
                        // программе параметров нет. В
// случае ее сбоя возвращается
                        // список сообщений.
     if (pgm.run() != true)
     {
                        // Переход к этому месту означает
                        // сбой программы. Выдается список
// сообщений, позволяющий определить
                        // причину ошибки.
        AS400Message[] messageList = pgm.getMessageList();

                        // ... Обработка списка сообщений.
     }

                        // Отключение после завершения
                        // программ
     sys.disconnectService(AS400.COMMAND);
```

В случае применения объекта ProgramCall должен быть указан полный путь в интегрированной файловой системе к программе.

R The default behavior is for System i5 programs to run in a separate server job, even when the Java program and the
R System i5 program are on the same server. You can override the default behavior and have the System i5 program run
R in the Java job using the ProgramCall setThreadSafe() method.

Применение объектов ProgramParameter

R You can use the ProgramParameter objects to pass parameter data between the Java program and the System i5
R program. Set the input data with the setInputData() method. After the program is run, retrieve the output data with the
R getOutputData() method. Каждый параметр представляет собой массив байтов. The Java program must convert
R the byte array between Java and System i5 formats. Методы преобразования данных содержатся в классах
R преобразования данных. Параметры добавляются в объект ProgramCall в виде списка.

Пример: Применение класса ProgramParameter

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведен пример использования объекта ProgramParameter для передачи параметров.

```
R                       // Создание объекта AS400
R     AS400 sys = new AS400("mySystem.myCompany.com");
R
R                       // Данная программа использует два параметра.
R                       // Для хранения этих параметров создается
R                       // список.
R     ProgramParameter[] parmList = new ProgramParameter[2];
R
R                       // Первый параметр -
R                       // входной
R     byte[] key = {1, 2, 3};
R     parmList[0] = new ProgramParameter(key);
R
R                       // Второй параметр - выходной.
R                       // параметра. Возвращается
```

```
R                       // четырехбайтовое значение.
R      parmList[1] = new ProgramParameter(4);
R
R                       // Создание объекта программы.
R                       // Задаются имя программы и список
R                       // параметров.
R      ProgramCall pgm = new ProgramCall(sys, "/QSYS.LIB/MYLIB.LIB/MYPROG.PGM", parmList);
R
R                       // Запуск программы.
R      if (pgm.run() != true)
R      {
R
R                       // If the system cannot run the
R                       // программу, просмотрите список сообщений
R                       // и найдите причину ошибки.
R         AS400Message[] messageList = pgm.getMessageList();
R
R      }
R      else
R      {
R                       // Программа выполнена. Обрабатывается
R // второй параметр, содержащий
R                       // возвращаемые данные.
R
R                       // Объект-преобразователь
R                       // System i5 data type
R         AS400Bin4 bin4Converter = new AS400Bin4();
R
R                       // Convert from system type to Java
R                       // Число расположено в начале
R                       // буфера.
R         byte[] data = parmList[1].getOutputData();
R         int i = bin4Converter.toInt(data);
R      }
R
R                       // Отключение после завершения
R                       // программ
R      sys.disconnectService(AS400.COMMAND);
```

ProgramCall Javadoc

ProgramParameter Javadoc

AS400Message Javadoc

AS400 Javadoc

CommandCall Javadoc

ProgramParameter Javadoc

Класс QSYSObjectPathName

You can use the QSYSObjectPathName class to represent an object in the integrated file system. Его можно применять для формирования имени объекта в интегрированной файловой системе и для разбиения этого имени на составные части при синтаксическом анализе.

Некоторые классы IBM Toolbox for Java требуют указывать полное имя в интегрированной файловой системе. Для конструирования этого имени можно применять объект QSYSObjectPathName.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведены примеры использования класса QSYSObjectPathName:

Пример 1: В объекте ProgramCall необходимо задать полное имя вызываемой программы сервера в интегрированной файловой системе. Для конструирования имени применяется объект QSYSObjectPathName. Код вызова программы PRINT_IT, находящейся в библиотеке REPORTS, с помощью объекта QSYSObjectPathName выглядит следующим образом:

```
                    // Создание объекта AS400.      AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта вызова программы.
ProgramCall pgm = new ProgramCall(sys);

                    // Создание объекта полного имени
                    // программы PRINT_IT из
                    // библиотеки REPORTS.
QSYSObjectPathName pgmName = new QSYSObjectPathName("REPORTS",
                                                    "PRINT_IT",
                                                    "PGM");

                    // Объект полного имени применяется
                    // для указания имени объекта вызова
                    // программы.
pgm.setProgram(pgmName.getPath());

                    // ... выполнение программы,
                    // обработка результатов
```

Example 2: If the name of the object is used just once, the Java program can use the toPath() method to build the name. Это эффективнее, чем создание имени с помощью объекта QSYSObjectPathName.

```
                    // Создание объекта AS400.      AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта вызова программы.
ProgramCall pgm = new ProgramCall(sys);

                    // Создание полного имени программы
                    // PRINT_IT из библиотеки REPORTS
                    // с помощью метода toPath.
pgm.setProgram(QSYSObjectPathName.toPath("REPORTS",
                                         "PRINT_IT",
                                         "PGM"));

                    // ... выполнение программы,
                    // обработка результатов
```

Пример 3: В этом примере предполагается, что в программу на Java был передан путь в интегрированной файловой системе. The QSYSObjectPathName class can be used to parse this name into its components.

```
                    // Создание объекта полного имени
                    // из полного пути в интегрированной
                    // файловой системе.
QSYSObjectPathName ifsName = new QSYSObjectPathName(pathName);

                    // Использование объекта полного имени
                    // для получения библиотеки, имени и типа
                    // объекта сервера.
String library = ifsName.getLibraryName();
String name    = ifsName.getObjectName();
String type    = ifsName.getObjectType();
```

QSYSObjectPathName Javadoc

Доступ на уровне записей

The record-level access classes create, read, update, and delete System i5 files and members.

R • Create a System i5 physical file specifying one of the following:

R – Длину записи

R – Исходный файл спецификаций описаний существующих данных (DDS)

R – Объект RecordFormat

R • Retrieve the record format from a physical or logical file, or the record formats from a System i5 multiple format
R logical file.

R **Примечание:** Сведения о формате записей передаются не полностью. Получаемая информация о
R форматах записей предназначается для настройки формата записи для объекта AS400File.
R Запрашивается только та информация, которая необходима для описания содержимого
R записи в файле. Например, информация о заголовках колонок и о псевдонимах для этого не
R нужна, поэтому она не передается.

R • Access the records in an System i5 file sequentially, by record number, or by key.

R • Write records to a system file.

R • Update records in a system file sequentially, by record number, or by key.

R • Delete records in a system file sequentially, by record number, or by key.

R • Lock a file for different types of access.

R • Устанавливать режим управления фиксацией, позволяющий программе на Java выполнять следующие
R операции:

R – Включать для соединения опцию управления фиксацией.

R – Устанавливать для разных файлов различные уровни блокировки.

R – Выполнять фиксацию и откат транзакций.

R • Delete system files.

R • Delete a member from a system file.

Примечание: Классы доступа уровня записи не поддерживают логическое объединение файлов и пустые поля ключей.

Классы доступа на уровне записей реализуют следующие функции:

- AS400File - это абстрактный базовый класс для классов доступа на уровне записей. Он включает методы последовательного доступа к записям, создания и удаления файлов и их элементов, а также управления фиксацией.
- The KeyedFile class represents a system file that has access by key..
- The SequentialFile class represents a system file that has access by record number.
- The AS400FileRecordDescription class provides the methods for retrieving the record format of a system file.

Классы доступа на уровне записей требуют, чтобы был создан объект AS400, соответствующий той системе, в которой находятся файлы базы данных. Using the record-level access classes causes the AS400 object to connect to the System i. Информация об управлении соединениями приведена в разделе управление соединениями.

Классы доступа на уровне записей требуют указывать полное имя файла базы данных (путь в интегрированной файловой системе). Дополнительная информация приведена в разделе Пути к объектам IFS.

Классы доступа на уровне записей используют следующие классы:

- Класс RecordFormat - для описания записи в файле базы данных
- Класс Record - для предоставления доступа к записям в файле базы данных
- Класс LineDataRecordWriter - для добавления записи в формате строковых данных

Эти классы описаны в разделе Преобразование данных.

Примеры

R • The sequential access example shows how to access a system file sequentially.

R • The read file example shows how to use the record-level access classes to read a system file.

R • The keyed file example shows how to use the record-level access classes to read records by key from a system file.

Класс AS400File:

The AS400File class provides the methods for several actions.

- Создание и удаление физических файлов и элементов сервера
- Чтение и добавление записей в файлы сервера
- Блокирование доступа к файлам (для различных типов прав доступа)
- Объединение записей в блоки для повышения производительности
- Выбор позиции курсора в открытом файле сервера
- Управление фиксацией

Информация, связанная с данной

AS400File Javadoc

Класс KeyedFile:

The KeyedFile class gives a Java program keyed access to a file on the server. Доступом по ключу называется режим доступа программы на Java, при котором записи в файлах упорядочены по специальным значениям - ключам. Класс содержит методы для установки курсора, чтения, обновления и удаления записей по ключу.

Для установки курсора предназначены следующие методы:

- positionCursor(Object[]) - set cursor to the first record with the specified key.
- positionCursorAfter(Object[]) - set cursor to the record after the first record with the specified key.
- positionCursorBefore(Object[]) - set cursor to the record before the first record with the specified key.

Для удаления записи предназначен следующий метод:

- deleteRecord(Object[]) - delete the first record with the specified key.

Для чтения записи предназначены следующие методы:

- read(Object[]) - read the first record with the specified key.
- readAfter(Object[]) - read the record after the first record with the specified key.
- readBefore(Object[]) - read the record before the first record with the specified key.
- readNextEqual() - read the next record whose key matches the specified key. Поиск начинается с записи, которая находится после текущей позиции курсора.
- readPreviousEqual() - read the previous record whose key matches the specified key. Поиск начинается с записи, которая находится перед текущей позицией курсора.

Для обновления записи предназначен следующий метод:

- update(Object[]) - update the record with the specified key.

Класс также содержит методы для указания критерия поиска при установке курсора, чтении или обновлении по ключу. Допустимы следующие значения критерия поиска:

- Equal - find the first record whose key matches the specified key.
- Less than - find the last record whose key comes before the specified key in the key order of the file.
- Less than or equal - find the first record whose key matches the specified key. Если такие записи отсутствуют, выполняется поиск последней записи, ключ которой расположен перед указанным ключом в последовательности ключей файла.
- Greater than - find the first record whose key comes after the specified key in the key order of the file.

- Greater than or equal - find the first record whose key matches the specified key. Если такие записи отсутствуют, выполняется поиск первой записи, ключ которой расположен после указанного ключа в последовательности ключей файла.

KeyedFile является подклассом класса AS400File; все методы класса AS400File доступны в классе KeyedFile.

Указание ключа

Ключ для объекта KeyedFile представляет собой массив объектов Java, типы и порядок расположения которых соответствуют типам и последовательности ключевых полей, определяемым для файла объектом RecordFormat.

Ниже приведен пример указания ключа для объекта KeyedFile.

```
            // Указание ключа для файла со следующей
            // последовательностью ключевых полей:
            //    CUSTNAME    CHAR(10)
            //    CUSTNUM     BINARY(9)
            //    CUSTADDR    CHAR(100)VARLEN()
            // Заметьте, что последнее поле - переменной длины.
    Object[] theKey = new Object[3];
    theKey[0] = "John Doe";
    theKey[1] = new Integer(445123);
    theKey[2] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

Объект KeyedFile принимает как полные, так и неполные ключи. Однако при указании значений ключевых полей необходимо следить за их порядком.

Например:

```
            // Указание неполного ключа для файла со следующей
            // последовательностью ключевых полей:
            //    CUSTNAME    CHAR(10)
            //    CUSTNUM     BINARY(9)
            //    CUSTADDR    CHAR(100)VARLEN()
    Object[] partialKey = new Object[2];
    partialKey[0] = "John Doe";
    partialKey[1] = new Integer(445123);

            // Пример НЕПРАВИЛЬНОГО неполного ключа
    Object[] INVALIDPartialKey = new Object[2];
    INVALIDPartialKey[0] = new Integer(445123);
    INVALIDPartialKey[1] = "2227 John Doe Lane, ANYTOWN, NY 11199";
```

Пустые ключи и ключевые поля не поддерживаются.

The key field values for a record can be obtained from the Record object for a file through the getKeyFields() method.

Ниже приведен пример чтения записей из файла по ключу:

```
            // Создайте объект AS400; файл
            // находится на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");

            // Создайте файловый объект для представления файла.
    KeyedFile myFile = new KeyedFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
            // Предполагается, что для генерации кода для подкласса
            // RecordFormat, представляющего формат записи
            // файла MYFILE в библиотеке MYLIB, применялся
            // класс AS400FileRecordDescription. Этот код был
// откомпилирован, и теперь его можно использовать в программе на Java.
    RecordFormat recordFormat = new MYKEYEDFILEFormat();

            // Задайте формат записи для файла myFile. Это
```

```
// необходимо сделать до вызова метода open()
     myFile.setRecordFormat(recordFormat);

             // Откройте файл
     myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

             // Формат записи для файла содержит четыре
             // ключевых поля в следующей последовательности:
             // CUSTNUM, CUSTNAME, PARTNUM и ORDNUM.
             // Неполный ключ partialKey будет содержать значения
             // двух ключевых полей. Так как при указании значений
// ключевых полей учитывается порядок, partialKey
             // будет содержать значения для CUSTNUM и CUSTNAME.
     Object[] partialKey = new Object[2];
     partialKey[0] = new Integer(1);
     partialKey[1] = "John Doe";

             // Чтение первой совпадающей с ключом partialKey записи
     Record keyedRecord = myFile.read(partialKey);

             // Если запись не найдена, возвращается null.
     if (keyedRecord != null)
     { // Запись для John Doe найдена, информация выводится на печать.
       System.out.println("Информация для заказчика " + (String)partialKey[1] + ":");
       System.out.println(keyedRecord);
     }

                     ....

             // После завершения работы с файлом его необходимо закрыть
     myFile.close();

             // После выполнения операций с доступом на уровне
             // записей соединение необходимо закрыть
     sys.disconnectService(AS400.RECORDACCESS);
```

Информация, связанная с данной

KeyedFile Javadoc

Класс SequentialFile:

The SequentialFile class gives a Java program access to a file on the server by record number. Класс содержит методы для установки курсора, чтения, обновления и удаления записи по ее номеру.

Для установки курсора предназначены следующие методы:

- positionCursor(int) - set cursor to the record with the specified record number.
- positionCursorAfter(int) - set cursor to the record after the specified record number.
- positionCursorBefore(int) - set cursor to the record before the specified record number.

Для удаления записи предназначен следующий метод:

- deleteRecord(int) - delete the record with the specified record number.

Для чтения записи предназначены следующие методы:

- read(int) - read the record with the specified record number.
- readAfter(int) - read the record after the specified record number.
- readBefore(int) - read the record before the specified record number.

Для обновления записи предназначен следующий метод:

- update(int) - update the record with the specified record number.

SequentialFile является подклассом класса AS400File; все методы AS400File доступны для SequentialFile.

Ниже приведен пример использования класса SequentialFile:

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте файловый объект для представления файла.
     SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                    // Предполагается, что для генерации кода для подкласса
                    // RecordFormat, представляющего формат записи
                    // файла MYFILE в библиотеке MYLIB, применялся
                    // класс AS400FileRecordDescription. Этот код был
// откомпилирован, и теперь его можно использовать в программе на Java.
     RecordFormat recordFormat = new MYFILEFormat();

                    // Задайте формат записи для файла myFile. Это
// необходимо сделать до вызова метода open()
     myFile.setRecordFormat(recordFormat);

                    // Откройте файл
     myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Удалите запись номер 2.
     myFile.delete(2);

                    // Прочитайте запись номер 5 и обновите ее
     Record updateRec = myFile.read(5);
     updateRec.setField("CUSTNAME", newName);

                    // Так как указатель уже находится на нужной записи,
                    // можно использовать метод update() базового класса.
     myFile.update(updateRec);

                    // Обновление записи номер 7
     updateRec.setField("CUSTNAME", nextNewName);
     updateRec.setField("CUSTNUM", new Integer(7));
     myFile.update(7, updateRec);

                   ....

                    // После завершения работы с файлом его необходимо закрыть
     myFile.close();

                    // После выполнения операций с доступом на уровне
                    // записей соединение необходимо закрыть
     sys.disconnectService(AS400.RECORDACCESS);
```

Информация, связанная с данной

SequentialFile Javadoc

Класс AS400FileRecordDescription:

The IBM Toolbox for Java AS400FileRecordDescription class provides the methods for retrieving the record format of a file on the server.

This class provides methods for creating Java source code for subclasses of RecordFormat and for returning RecordFormat objects, which describe the record formats of user-specified physical or logical files on the server. Значения, возвращаемые этими методами, могут использоваться в качестве входных данных при настройке формата записи для объекта AS400File.

Если файл уже существует на сервере, для создания объекта RecordFormat рекомендуется всегда применять класс AS400FileRecordDescription.

Примечание: Класс AS400FileRecordDescription получает не полный формат записи файла, а только информацию о содержимом записей. Такая информация, как, например, заголовки колонок, псевдонимы и поля ссылок, не передается. Поэтому полученные форматы записей нельзя применять для создания другого файла того же формата.

Создание исходного кода Java для подклассов RecordFormat, представляющих формат записи файлов сервера

The AS400FileRecordDescription createRecordFormatSource() method creates Java source files for subclasses of the RecordFormat class. The files can be compiled and used by an application or applet as input to the AS400File.setRecordFormat() method.

Метод createRecordFormatSource() позволяет узнавать форматы записей файлов сервера на этапе создания исходного кода. С помощью этого метода создается исходный код для подклассов класса RecordFormat, который может применяться различными программами на Java для обращения к одним и тем же файлам сервера. При необходимости этот код можно изменить. Поскольку данный метод создает файлы в локальной системе, он может использоваться только приложениями на Java. Однако данные, возвращаемые этим методом (исходный код на Java), можно откомпилировать и затем использовать как в приложениях, так и в аплетах Java.

Примечание: Этот метод заменяет файлы, имена которых совпадают с именами созданных исходных файлов Java.

Пример 1: Использование метода createRecordFormatSource():

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта AS400FileRecordDescription для представления файла
    AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys,
                                        "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");
// Создание файла с исходным кодом на Java в текущем рабочем каталоге.
                    // Укажите "package com.myCompany.myProduct;" для оператора
                    // package в исходном файле, поскольку класс поставляется как
                    // компонент продукта myProduct.
    myFile.createRecordFormatSource(null, "com.myCompany.myProduct");

                    // Предполагается, что для файла MYFILE имя формата -  FILE1; тогда
                    // в текущем рабочем каталоге будет создан файл FILE1Format.java.
                    // Если файл с таким именем существует, он будет заменен на вновь созданный.
// Классу присваивается имя FILE1Format. Это производный класс от RecordFormat.
```

Пример 2: Откомпилируйте файл, созданный в предыдущем примере, и используйте его следующим образом:

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте объект AS400File для файла
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Задание формата записи
                    // оператор import.com.myCompany.myProduct.FILE1Format;
                    // был указан ранее.

    myFile.setRecordFormat(new FILE1Format());

                    // Открытие файла и чтение данных
                    ....

                    // После завершения работы с файлом его необходимо закрыть
    myFile.close();
```

```
                    // После выполнения операций с доступом на уровне
                    // записей соединение необходимо закрыть
     sys.disconnectService(AS400.RECORDACCESS);
```

Создание объектов RecordFormat, представляющих формат записей файлов сервера

The AS400FileRecordDescription retrieveRecordFormat() method returns an array of RecordFormat objects that represent the record formats of an existing file on the server. Обычно этот массив состоит из одного объекта RecordFormat. Однако в случае логического файла с несколькими форматами этот массив содержит несколько объектов RecordFormat. Этот метод позволяет узнать формат существующего файла сервера во время работы программы. The RecordFormat object then can be used as input to the AS400File.setRecordFormat() method.

Ниже приведен пример использования метода retrieveRecordFormat():

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта AS400FileRecordDescription для представления файла
     AS400FileRecordDescription myFile = new AS400FileRecordDescription(sys,
                                         "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");
// Получение информации о формате записи для этого файла
     RecordFormat[] format = myFile.retrieveRecordFormat();

                    // Создайте объект AS400File для файла
     SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE");

                    // Задание формата записи
     myFile.setRecordFormat(format[0]);

                    // Открытие файла и чтение данных
                    ....

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

                    // После выполнения операций с доступом на уровне
                    // записей соединение необходимо закрыть
     sys.disconnectService(AS400.RECORDACCESS);
```

AS400FileRecordDescription Javadoc

RecordFormat Javadoc

AS400File Javadoc

Создание и удаление файлов и элементов:

Physical files on the server are created by specifying a record length, an existing server data description specifications (DDS) source file, or a RecordFormat object.

Если при создании файла вы указываете длину записи, может быть создан файл данных или исходный файл. Формат записи для объекта задается методом. Не вызывайте для объекта метод setRecordFormat().

Файл данных содержит одно поле. Имя поля совпадает с именем файла, тип поля - символьный, длина поля указывается как параметр в методе create.

Исходный файл содержит три поля:

- Поле SRCSEQ в зонном десятичном формате (6,2)
- Поле SRCDAT в зонном десятичном формате (6,0)
- Поле SRCDTA - символьное поле, длина которого равна значению, указанному в методе create, минус 12.

Ниже приведены примеры создания файлов и элементов.

Пример 1: Создание файла данных с записями длиной 128 байт:

```
                    // Создайте объект AS400; файл будет создан
                    // на этом сервере.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте файловый объект для представления файла.
     SequentialFile newFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                    // Создайте файл
     newFile.create(128, "*DATA", "Файл данных с длиной записи 128 байт");

                    // Откройте файл только для записи.
                    // Примечание: Формат записи для файла
                    // уже задан в методе create()
     newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Занесите запись в файл. Поскольку формат записи
// задавался в create(), для получения правильно
                    // форматированной записи из этого файла можно
                    // использовать метод getRecordFormat().
     Record writeRec = newFile.getRecordFormat().getNewRecord();
     writeRec.setField(0, "Запись 1");
     newFile.write(writeRec);

                   ....

                    // После завершения работы с файлом его необходимо закрыть
     newFile.close();
                    // Завершите соединение, отключив доступ
                    // на уровне записей
     sys.disconnectService(AS400.RECORDACCESS);
```

Пример 2: При создании файла с помощью указания существующего исходного файла DDS последний задается как параметр в методе create(). The record format for the file must be set using the setRecordFormat() method before the file can be opened. Например:

```
                    // Создайте объект AS400; файл будет создан
                    // на этом сервере.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте объекты QSYSObjectPathName
                    // для нового файла и для файла DDS.
     QSYSObjectPathName file    = new QSYSObjectPathName("MYLIB", "MYFILE", "FILE", "MBR");
     QSYSObjectPathName ddsFile = new QSYSObjectPathName("MYLIB", "DDSFILE", "FILE", "MBR");

                    // Создайте файловый объект для представления файла.
     SequentialFile newFile = new SequentialFile(sys, file);

                    // Создайте файл
     newFile.create(ddsFile,  "Файл, создаваемый с помощью описания DDSFile");

                    // Задайте формат записи для файла,
                                          // получив его с сервера.
     newFile.setRecordFormat(new AS400FileRecordDescription(sys,
     newFile.getPath()).retrieveRecordFormat()[0]);

                    // Откройте файл для записи
     newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Занесите запись в файл. Для получения записи
// по умолчанию для файла применяется метод
               // getRecordFormat(), а затем - метод getNewRecord().
     Record writeRec = newFile.getRecordFormat().getNewRecord();
     newFile.write(writeRec);
```

```
                    ....

                      // После завершения работы с файлом его необходимо закрыть
     newFile.close();
                      // Завершите соединение, отключив доступ
                      // на уровне записей
     sys.disconnectService(AS400.RECORDACCESS);
```

Пример 3: При создании файла с помощью объекта RecordFormat последний задается в методе create(). Формат записи для объекта задается методом. Не вызывайте для объекта метод setRecordFormat().

```
                      // Создайте объект AS400; файл будет создан
                      // на этом сервере.
     AS400 sys = new AS400("mySystem.myCompany.com");

                      // Создайте файловый объект для представления файла.
     SequentialFile newFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                     // Получите формат записи из существующего файла
     RecordFormat recordFormat = new AS400FileRecordDescription(sys,
     "/QSYS.LIB/MYLIB.LIB/EXISTING.FILE/MBR1.MBR").retrieveRecordFormat()[0];

                      // Создайте файл
     newFile.create(recordFormat,  "Файл, создаваемый с помощью объекта формата записи");

                      // Откройте файл только для записи.
                      // Примечание: Формат записи для файла
                      // уже задан в методе create()
     newFile.open(AS400File.WRITE_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                      // Занесите запись в файл. Для получения
// записи по умолчанию в правильном формате
                      // используется recordFormat.
     Record writeRec = recordFormat.getNewRecord();
     newFile.write(writeRec);

                    ....

                      // После завершения работы с файлом его необходимо закрыть
     newFile.close();
                      // Завершите соединение, отключив доступ
                      // на уровне записей
     sys.disconnectService(AS400.RECORDACCESS);
```

Для удаления файлов и элементов предназначены следующие методы:

- Use the delete() method to delete server files and all of their members.
- Use the deleteMember()method to delete just one member of a file.

Use the addPhysicalFileMember() method to add members to a file.

Информация, связанная с данной

AS400File Javadoc

Чтение и сохранение записей:

You can use the AS400File class to read, write, update, and delete records in files on the server.

Для доступа к записи используется класс Record, который описывается классом RecordFormat. The record format must be set through the setRecordFormat() method before the file is opened, unless the file was just created (without an intervening close()) by one of the create() methods, which sets the record format for the object.

Для чтения записи из файла предназначены методы read(). Эти методы выполняют следующие операции:

- read() - read the record at the current cursor position
- readFirst() - read the first record of the file
- readLast() - read the last record of the file
- readNext() - read the next record in the file
- readPrevious() - read the previous record in the file

Ниже приведен пример использования метода readNext():

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте файловый объект для представления файла.
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                    // Предполагается, что для генерации кода для подкласса
                    // RecordFormat, представляющего формат записи
                    // файла MYFILE в библиотеке MYLIB, применялся
                    // класс AS400FileRecordDescription. Этот код был
// откомпилирован, и теперь его можно использовать
                    // в программе на Java.
    RecordFormat recordFormat = new MYFILEFormat();

                    // Задайте формат записи для файла myFile. Это
// необходимо сделать до вызова метода open()
    myFile.setRecordFormat(recordFormat);

                    // Откройте файл
    myFile.open(AS400File.READ_ONLY, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Прочитайте все записи в файле, занося значение
                    // поля CUSTNAME ("Имя заказчика") в файл System.out
    System.out.println("             СПИСОК ЗАКАЗЧИКОВ");
    System.out.println("_____________________________________________");

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

                  ....

                    // После завершения работы с файлом его необходимо закрыть
    myFile.close();

                    // Завершите соединение, отключив доступ
                    // на уровне записей
    sys.disconnectService(AS400.RECORDACCESS);
```

Use the update() method to update the record at the cursor position.

Например:

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте файловый объект для представления файла.
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                    // Предполагается, что для генерации кода для подкласса
                    // RecordFormat, представляющего формат записи
                    // файла MYFILE в библиотеке MYLIB, применялся
```

```
                    // класс AS400FileRecordDescription. Этот код был
// откомпилирован, и теперь его можно использовать в программе на Java.
     RecordFormat recordFormat = new MYFILEFormat();

                    // Задайте формат записи для файла myFile. Это
// необходимо сделать до вызова функции open()
     myFile.setRecordFormat(recordFormat);

                    // Откройте файл для обновления
     myFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Обновите первую запись в файле. Предполагается,
// что newName - это строка с новым именем для
                    // CUSTNAME
     Record updateRec = myFile.readFirst();
     updateRec.setField("CUSTNAME", newName);
     myFile.update(updateRec);

                   ....

                    // После завершения работы с файлом его необходимо закрыть
     myFile.close();

                    // После выполнения операций с доступом на уровне
                    // записей соединение необходимо закрыть
     sys.disconnectService(AS400.RECORDACCESS);
```

Use the write() method to append records to the end of a file. В файл можно добавлять одну запись или массив записей.

Use the deleteCurrentRecord() method to delete the record at the cursor position.

Информация, связанная с данной

AS400File Javadoc

Блокирование файлов:

The Java program can lock a file to prevent other users from accessing the file while the first Java program is using the file.

Lock types are as follows. You can find more information about the lock types in the AS400File Javadoc.

- Read/Exclusive Lock - The current Java program reads records, and no other program can access the file.
- Read/Allow shared read Lock - The current Java program reads records, and other programs can read records from the file.
- Read/Allow shared write Lock - The current Java program reads records, and other programs can change the file.
- Write/Exclusive Lock - The current Java program changes the file, and no other program can access the file.
- Write/Allow shared read Lock - The current Java program changes the file, and other programs can read records from the file.
- Write/Allow shared write Lock - The current Java program changes the file, and other programs can change the file.

To give up the locks obtained through the lock() method, the Java program starts the releaseExplicitLocks() method.

Информация, связанная с данной

AS400File Javadoc

Объединение записей в блоки:

The IBM Toolbox for Java AS400File class uses record blocking to improve performance.

- Если файл открыт только для чтения, то при считывании записи программой на Java считывается блок записей. Blocking improves performance because subsequent read requests may be handled without accessing the

server. На считывание нескольких записей тратится лишь немногим больше времени, чем на считывание одной записи. Быстродействие значительно возрастает, если записи можно хранить в виде блоков в кэше на компьютере-клиенте.

Число записей в блоке можно задать при открытии файла. Например:

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте файловый объект для представления файла.
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                    // Предполагается, что для генерации кода для подкласса
                    // RecordFormat, представляющего формат записи
                    // файла MYFILE в библиотеке MYLIB, применялся
                    // класс AS400FileRecordDescription. Этот код был
// откомпилирован, и теперь его можно использовать
                    // в программе на Java.
    RecordFormat recordFormat = new MYFILEFormat();

                    // Задайте формат записи для файла myFile. Это
// необходимо сделать до вызова метода open()
    myFile.setRecordFormat(recordFormat);

                    // Откройте файл. Укажите число записей в блоке, равное 50:
int blockingFactor = 50;
    myFile.open(AS400File.READ_ONLY, blockingFactor, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Чтение первой записи из файла.  Поскольку
                    // было задано число записей в блоке, при вызове
                    // функции read() будет считано 50 записей.
    Record record = myFile.readFirst();
    for (int i = 1; i < 50 && record != null; i++)
    {

      // Записи, считанные в этом цикле, будут помещены в виде блока
      // в кэш клиента.
      record = myFile.readNext();
    }
                  ....

                    // После завершения работы с файлом его необходимо закрыть
    myFile.close();

                    // Завершите соединение, отключив доступ
                    // на уровне записей
    sys.disconnectService(AS400.RECORDACCESS);
```

- If the file is opened for write-only access, the blocking factor indicates how many records are written to the file at one time when the write(Record[]) method is invoked.

  Например:

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте файловый объект для представления файла.
    SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");

                    // Предполагается, что для генерации кода для подкласса
                    // RecordFormat, представляющего формат записи
                    // файла MYFILE в библиотеке MYLIB, применялся
                    // класс AS400FileRecordDescription. Этот код был
// откомпилирован, и теперь его можно использовать
                    // в программе на Java.
    RecordFormat recordFormat = new MYFILEFormat();
```

```
                    // Задайте формат записи для файла myFile. Это
// необходимо сделать до вызова функции open()
     myFile.setRecordFormat(recordFormat);

                    // Откройте файл. Укажите число записей в блоке, равное 50:
int blockingFactor = 50;
     myFile.open(AS400File.WRITE_ONLY, blockingFactor, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Создайте массив записей, которые будут заноситься в файл
     Record[] records = new Record[100];
     for (int i = 0; i < 100; i++)
     {
                    // Предположим, что в файле есть два поля:
                    // ИМЯ_ЗАКАЗЧИКА и НОМЕР_ЗАКАЗЧИКА
       records[i] = recordFormat.getNewRecord();
       records[i].setField("ИМЯ_ЗАКАЗЧИКА", "Заказчик " + String.valueOf(i));
       records[i].setField("НОМЕР_ЗАКАЗЧИКА", new Integer(i));
     }

                    // Занесите записи в файл. Поскольку число записей
// в блоке равно 50, цикл сохранения записей включает
                    // только 2 шага - на каждом шаге сохраняется 50 записей
     myFile.write(records);

                   ....

                    // После завершения работы с файлом его необходимо закрыть
     myFile.close();

                    // Завершите соединение, отключив доступ
                    // на уровне записей
     sys.disconnectService(AS400.RECORDACCESS);
```

- Если файл открыт для чтения и для записи, то объединение записей в блоки не применяется. Если при вызове метода open() задано число записей в блоке, оно игнорируется.

  AS400File Javadoc

Задание позиции курсора:

В любом открытом файле есть курсор. Курсор указывает на читаемую, обновляемую или удаляемую запись. Если файл открывается в первый раз, курсор указывает на начало файла, т.е. располагается перед первой записью в файле.

Для задания позиции курсора предназначены следующие методы:

- positionCursorAfterLast() - Set cursor to after the last record. Если применяется данный метод, то для обращения к записям в файле программа на Java может использовать метод readPrevious().
- positionCursorBeforeFirst() - Set cursor to before the first record. Если применяется данный метод, то для обращения к записям в файле программа на Java может использовать метод readNext().
- positionCursorToFirst() - Set the cursor to the first record.
- positionCursorToLast() - Set the cursor to the last record.
- positionCursorToNext() - Move the cursor to the next record.
- positionCursorToPrevious() - Move the cursor to the previous record.

Ниже приведен пример установки курсора с помощью метода positionCursorToFirst().

```
                    // Создайте объект AS400; файл
                    // находится на сервере.
     AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создайте файловый объект для представления файла.
     SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
```

```
                    // Предполагается, что для генерации кода для подкласса
                    // RecordFormat, представляющего формат записи
                    // файла MYFILE в библиотеке MYLIB, применялся
                    // класс AS400FileRecordDescription. Этот код был
// откомпилирован, и теперь его можно использовать
                    // в программе на Java.
    RecordFormat recordFormat = new MYFILEFormat();

                    // Задайте формат записи для файла myFile. Это
// необходимо сделать до вызова метода open()
    myFile.setRecordFormat(recordFormat);

                    // Открытие файла
    myFile.open(AS400File.READ_WRITE, 1, AS400File.COMMIT_LOCK_LEVEL_NONE);

                    // Удалите первую запись в файле:
    myFile.positionCursorToFirst();
    myFile.deleteCurrentRecord();

                  ....

                    // После завершения работы с файлом его необходимо закрыть
    myFile.close();

                    // Завершите соединение, отключив доступ
                    // на уровне записей
    sys.disconnectService(AS400.RECORDACCESS);
```

Информация, связанная с данной

AS400File Javadoc

Управление фиксацией:

Опция управления фиксацией позволяет программе на Java установить другой режим управления изменением файла. Если включен режим управления фиксацией, то транзакции в файле не выполняются, пока не произойдет их фиксация или откат. После фиксации изменения в файле становятся необратимыми. После отката изменения аннулируются. В зависимости от уровня управления фиксацией, установленного при обращении к методу open(), возможны следующие транзакции: изменение существующей записи, добавление записи, удаление записи и даже чтение записи.

The levels of commitment control are as follows. To see more details about each level, refer to the AS400File Javadoc.

- All - Every record accessed in the file is locked until the transaction is committed or rolled back.
- Change - Updated, added, and deleted records in the file are locked until the transaction is committed or rolled back.
- Cursor Stability - Updated, added, and deleted records in the file are locked until the transaction is committed or rolled back. Если приложение читает (но не изменяет) запись, то она остается блокированной только до тех пор, пока не произойдет обращение к другой записи.
- None - There is no commitment control on the file. Изменения в файле немедленно фиксируются и не могут быть аннулированы.

You can use the AS400File startCommitmentControl() method to start commitment control. При вызове этого метода для **соединения** с AS400 будет включен режим управления фиксацией, причем он будет распространяться на все файлы данного соединения, открытые после включения управления фиксацией. На файлы, открытые до этого момента, управление фиксацией не распространяется. The level of commitment control for individual files is specified on the open() method. Specify COMMIT_LOCK_LEVEL_DEFAULT to use the same level of commitment control as was specified on the startCommitmentControl() method.

Например:

```
                    // Создайте объект AS400; файлы
                    // находятся на сервере.
    AS400 sys = new AS400("mySystem.myCompany.com");
```

```
                    // Создайте три файловых объекта
SequentialFile myFile = new SequentialFile(sys, "/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/%FILE%.MBR");
SequentialFile yourFile = new SequentialFile(sys, "/QSYS.LIB/YOURLIB.LIB/YOURFILE.FILE/%FILE%.MBR");
SequentialFile ourFile = new SequentialFile(sys, "/QSYS.LIB/OURLIB.LIB/OURFILE.FILE/%FILE%.MBR");

                    // Перед включением режима управления фиксацией откройте файл yourFile.
                    // No commitment control applies to this file.                          // Поскольку управление фи
                    // параметр уровня блокировки фиксации игнорируется.
yourFile.setRecordFormat(new YOURFILEFormat());
yourFile.open(AS400File.READ_WRITE, 0, AS400File.COMMIT_LOCK_LEVEL_DEFAULT);

                    // Включите режим управления фиксацией для соединения.
                    // Примечание: для вызова метода startCommitmentControl() можно
                    // использовать любой из трех файлов.
myFile.startCommitmentControl(AS400File.COMMIT_LOCK_LEVEL_CHANGE);

                    // Откройте файлы myFile и ourFile
myFile.setRecordFormat(new MYFILEFormat());

                    // Укажите тот же уровень блокировки, который
                    // был задан при включении режима управления фиксацией
myFile.open(AS400File.WRITE_ONLY, 0, COMMIT_LOCK_LEVEL_DEFAULT);

ourFile.setRecordFormat(new OURFILEFormat());
                    // Укажите иной уровень блокировки, нежели тот,
                    // был задан при включении режима управления фиксацией
ourFile.open(AS400File.READ_WRITE, 0, COMMIT_LOCK_LEVEL_CURSOR_STABILITY);

                    // занесите записи в файлы (или обновите записи)
                   ....

                    // Зафиксируйте изменения в файлах myFile и ourFile.
                    // Заметьте, что функция commit фиксирует все изменения для соединения,
          // даже если вызывается только для одного объекта AS400File.
myFile.commit();
                    // Закройте файлы.
myFile.close();
yourFile.close();
ourFile.close();

                    // Отключите режим управления фиксацией.
          // При этом режим управления фиксацией для соединения будет отключен.
ourFile.endCommitmentControl();

                    // После выполнения операций с доступом на уровне
                    // записей соединение необходимо закрыть
sys.disconnectService(AS400.RECORDACCESS);
```

The commit() method commits all transactions since the last commit boundary for the **connection**. The rollback() method discards all transactions since the last commit boundary for the **connection**. Commitment control for a connection is ended through the endCommitmentControl() method. Если файл закрывается до того, как был вызван метод commit() или rollback(), все незафиксированные транзакции аннулируются. Все файлы, открытые после включения режима управления фиксацией, необходимо закрыть до вызова метода endCommitmentControl().

Приведенные ниже примеры иллюстрируют включение режима управления фиксацией, обращение к функциям commit и roll back и отключение режима.

```
                    // ... предполагается, что объект AS400 и файл
                    // реализованы.

                    // Включите режим управления фиксацией для уровня *CHANGE
aFile.startCommitmentControl(AS400File.COMMIT_LOCK_LEVEL_CHANGE);

                    // ... откройте файл и внесите в него изменения.
```

```
// Например, измените, добавьте или удалите записи.

                    // В зависимости от флага либо сохраните, либо
                    // аннулируйте транзакции.
   if (saveChanges)
      aFile.commit();
   else
      aFile.rollback();

                    // Закрыть файл
   aFile.close();

                    // Отключите режим управления фиксацией для соединения.
   aFile.endCommitmentControl();
```

Информация, связанная с данной

AS400File Javadoc

SaveFile class

Класс SaveFile отвечает за сохранение файла на сервере.

Информация, связанная с данной

SaveFile Javadoc

Вызовы служебных программ

The IBM Toolbox for Java ServiceProgramCall class allows you to call a System i service program. ServiceProgramCall is a subclass of the ProgramCall class that you use to call System i programs. If you want to call an System i program, use the ProgramCall class.

R The ServiceProgramCall class makes it possible for you to call an System i service program, pass data to a service
R program through input parameters, and access data the service program returns through output parameters. Using
R ServiceProgramCall causes the AS400 object to connect to the server. Информация об управлении соединениями
R приведена в разделе управление соединениями.

По умолчанию служебная программа запускается в отдельном задании сервера, даже если программа на Java и служебная программа выполняются на одном сервере. You can override the default behavior and have the service program run in the Java job using the setThreadSafe() method, inherited from ProgramCall.

Применение класса ServiceProgramCall

Перед началом работы с классом ServiceProgramCall убедитесь, что соблюдены следующие требования.

- The service program must be on an server
- Служебной программе передается не более семи параметров
- Служебная программа возвращает пустое или численное значение

Работа с объектами ProgramParameter

R The ProgramParameter class works with the ServiceProgramCall class to pass parameter data to and from a System i
R service program. You pass input data to the service program with setInputData().

You request the amount of output data you want returned with setOutputDataLength(). You retrieve the output data after the service program is finished running with getOutputData(). Помимо самих данных, в классе ServiceProgramCall требуется задать способ их передачи служебной программе. The setParameterType() method of ProgramParameter is used to supply this information. Атрибут type указывает, как передается параметр: по значению или по ссылке. В любом случае данные передаются от клиента серверу. Once the data is on the server, the server uses the parameter type to correctly call the service program.

Все параметры будут представлены в виде массива байт. Therefore, to convert between i5/OS and Java formats, you use the data conversion and description classes.

ServiceProgramCall Javadoc

ProgramCall Javadoc

ProgramParameter Javadoc

Subsystem class

Класс Subsystem представляет подсистему на сервере.

Информация, связанная с данной

Subsystem Javadoc

Классы SystemStatus

The SystemStatus classes allow you to retrieve system status information and to retrieve and change system pool information.

Объект SystemStatus содержит методы, позволяющие просматривать информацию о состоянии системы, в том числе:

- getUsersCurrentSignedOn(): Returns the number of users currently signed on the system
- getUsersTemporarilySignedOff(): Returns the number of interactive jobs that are disconnected
- getDateAndTimeStatusGathered(): Returns the date and time when the system status information was gathered
- getJobsInSystem(): Returns the total number of user and system jobs that are currently running
- getBatchJobsRunning(): Returns the number of batch jobs currently running on the system
- getBatchJobsEnding(): Returns the number of batch jobs that are in the process of ending
- getSystemPools(): Returns an enumeration containing a SystemPool object for each system pool

Класс SystemStatus позволяет обратиться не только к своим методам, но и к методам класса SystemPool. Класс SystemPool предназначен для просмотра и изменения параметров системного пула.

Пример

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

В приведенном ниже примере демонстрируется работа функции кэширования с классом SystemStatus:

```
AS400 system = new AS400("MyAS400");
SystemStatus status = new SystemStatus(system);

// Включение функции кэширования. По умолчанию она выключена.
status.setCaching(true);

// Получение значения из системы.
// Для всех последующих вызовов будет применяться
// значение из кэша, а не из системы.
int jobs = status.getJobsInSystem();


// ... Выполнение остальных операций ...


// Проверка, включена ли функция кэширования.
if (status.isCaching())
{
  // Получение значения из кэша.
  jobs = status.getJobsInSystem();
}

// Настройка на считывание значения из системы, независимо от его наличия в кэше.
status.refreshCache();
```

```
// Получение значения из системы.
jobs = status.getJobsInSystem();

// Выключение функции кэширования. Все последующие вызовы будут обращаться к системе.
status.setCaching(false);

// Получение значения из системы.
jobs = status.getJobsInSystem();
```

SystemStatus Javadoc

Класс SystemPool:

The SystemPool class allows you to retrieve and change system pool information.

The SystemPool class includes the following methods:

- The getPoolSize() method returns the size of the pool, and the setPoolSize() method sets the size of the pool.
- The getPoolName() method retrieves the name of the pool, and the setPoolName() method sets the pool's name.
- The getReservedSize() method returns the amount of storage in the pool that is reserved for system use.
- The getDescription() method returns the description of the system pool.
- The getMaximumActiveThreads() method returns the maximum number of threads that can be active in the pool at any one time.
- The setMaximumFaults() method sets the maximum faults-per-second guideline to use for this system pool.
- The setPriority() method sets the priority of this system pool relative to the priority of the other system pools.

Пример

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
//Создание объекта AS400.
AS400 as400 = new AS400("имя-системы");

//Создание объекта системного пула.
SystemPool systemPool = new SystemPool(as400,"*SPOOL");

//Получение опции подкачки системного пула
System.out.println("Опция подкачки : "+systemPool.getPagingOption());
```

Информация, связанная с данной

SystemPool Javadoc

Системные значения

The system value classes allow a Java program to retrieve and change system values and network attributes. You can also define your own group to contain the system values you want.

Объект SystemValue содержит следующую информацию:

- Name
- Description
- Release
- Value

Using the SystemValue class, retrieve a single system value by using the getValue() method and change a system value by using the setValue() method.

Кроме того, можно получить информацию о группе конкретного системного значения:

- To retrieve the system-defined group to which a system value belongs, use the getGroup() method.

- To retrieve the user-defined group to which a SystemValue object belongs (if any), use the getGroupName() and getGroupDescription() methods.

R Whenever the value of a system value is retrieved for the first time, the value is retrieved from the server and cached.
R При всех последующих обращениях к системному значению будет использоваться значение из кэша. If the
R current value is what you want instead of the cached value, a clear() must be done to clear the current cache.

Список системных значений

R SystemValueList represents a list of system values on the specified server. The list is divided into several
R system-defined groupsthat allow the Java program to access a portion of the system values at a time.

Группа системных значений

SystemValueGroup represents a user-defined collection of system values and network attributes. Этот класс предназначен не для хранения, а для создания и изменения наборов системных значений.

При создании объекта SystemValueGroup можно указать одну из системных групп (соответствующие константы заданы в классе SystemValueList) или список имен системных значений.

You can individually add the names of system values to include in the group by using the add() method. You can also remove them by using the remove() method.

Once the SystemValueGroup is populated with the required system value names, obtain the real SystemValue objects from the group by calling the getSystemValues() method. При этом объект SystemValueGroup создает из набора имен системных значений массив объектов SystemValue, в каждом из которых указывается имя системы, имя группы и описание группы, указанные в объекте SystemValueGroup.

To refresh a Vector of SystemValue objects all at once, use the refresh() method.

Примеры применения классов SystemValue и SystemValueList

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

В приведенном ниже примере создается и считывается системное значение:

```
// Создание объекта AS400
AS400 sys = new AS400("mySystem.myCompany.com");

//Создание системного значения, хранящего время в секундах.
SystemValue sysval = new SystemValue(sys, "QSECOND");

//Получение значения.
String second = (String)sysval.getValue();

//Значение QSECOND находится в кэше. Для получения текущего значения
//необходимо очистить кэш.
sysval.clear();
second = (String)sysval.getValue();

//Создание списка системных значений
SystemValueList list = new SystemValueList(sys);

//Получение всех системных значений, задающих дату и время.
Vector vec = list.getGroup(SystemValueList.GROUP_DATTIM);

//Отключение от системы.
sys.disconnectAllServices();
```

Примеры применения класса SystemValueGroup

Ниже приведен пример работы с пользовательской группой системных значений:

```
// Создание объекта AS400
AS400 sys = new AS400("mySystem.myCompany.com");

//Создание группы системных значений, включающей все сетевые атрибуты системы.
String name = "Группа";
String description = "Одно из системных значений.";
SystemValueGroup svGroup = new SystemValueGroup(sys, name, description, SystemValueList.GROUP_NET);

//Добавление в группу дополнительных системных значений и удаление лишних.
svGroup.add("QDATE");
svGroup.add("QTIME");
svGroup.remove("NETSERVER");
svGroup.remove("SYSNAME");

//Получение массива объектов SystemValue.
Vector sysvals = svGroup.getSystemValues();

//Вывод одного из системных значений.
SystemValue mySystemValue = (SystemValue)sysvals.elementAt(0);
System.out.println(mySystemValue.getName()+" - "+mySystemValue.getGroupDescription());

//Добавление в группу объекта SystemValue другой системы.
AS400 sys2 = new AS400("otherSystem.myCompany.com");
SystemValue sv = new SystemValue(sys2, "QDATE");
sysvals.addElement(sv);

//Одновременное обновление всей группы системных значений.
//It does not matter if some system values are from different System i5 servers.
//Способ создания системных значений (с помощью SystemValueGroup или нет) несущественен.
SystemValueGroup.refresh(sysvals);

//Отключение от систем.
sys.disconnectAllServices();
sys2.disconnectAllServices();
```

SystemValue Javadoc

SystemValueList Javadoc

SystemValueGroup Javadoc

Класс Trace

Класс Trace позволяет заносить в протокол точки трассировки и диагностические сообщения в программах на Java. Эта информация помогает воспроизводить неполадки и выполнять их диагностику.

Примечание: Можно также задать трассировку с помощью системных свойств трассировки.

Класс Trace регистрирует данные следующих категорий:

	Категория данных	**Описание**
	Преобразование	Регистрирует преобразование символов из кодовой страницы в формат Unicode и наоборот. Эта категория применяется только классами IBM Toolbox for Java.
R R R	Поток данных	Logs the data that flows between the system and the Java program. Эта категория применяется только классами IBM Toolbox for Java.
	Диагностика	Регистрирует информацию о состоянии.
	Ошибка	Регистрирует дополнительные ошибки, вызвавшие исключительную ситуацию.

Категория данных	Описание
Информация	Отслеживает поток данных в программе.
PCML	Эта категория применяется для определения способа интерпретации данных PCML, отправляемых на сервер и получаемых с сервера.
Сервер Proxy	Эта категория применяется классами IBM Toolbox for Java для сбора информации о данных, которыми обмениваются клиент и сервер Proxy.
Предупреждение	Регистрирует информацию об исправимых ошибках программы.
Все	Эта категория позволяет разрешить или запретить трассировку всех вышеперечисленных категорий. Данные трассировки для этой категории нельзя занести в протокол напрямую.

Классы IBM Toolbox for Java также применяют категории трассировки. При включении трассировки в программе на Java информация IBM Toolbox for Java регистрируется вместе с информацией приложения.

Вы можете выполнить трассировку для одной или нескольких категорий.. Once the categories are selected, use the setTraceOn method to turn tracing on and off. Data is written to the log using the log method.

Данные трассировки различных компонентов могут направляться в разные протоколы. Обычно данные трассировки записываются в протокол по умолчанию. Для записи данных трассировки приложения в другой протокол или стандартный вывод применяется трассировка компонентов. С ее помощью можно отделить данные трассировки приложения от остальных данных.

Чрезмерное занесение информации в протокол может снизить производительность. Use the isTraceOn method to query the current state of the trace. С помощью этого метода программа на Java определяет, нужно ли перед вызовом метода log создавать запись трассировки. Вызов метода log в то время, когда ведение протокола отключено, не приводит к ошибке, но снижает производительность.

По умолчанию информация протокола записывается в стандартный вывод. To redirect the log to a file, call the setFileName() method from your Java application. В общем случае это возможно только для приложений Java, так как большинство браузеров не позволяют аплетам записывать информацию в локальную файловую систему.

По умолчанию ведение протокола отключено. В программах на Java должна быть предусмотрена возможность включить ведение протокола. Например, в приложении можно определить параметр командной строки, задающий категории заносимых в протокол данных. В этом случае пользователь сможет задать этот параметр, как только ему потребуется собрать некоторую информацию.

Примеры

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведены примеры использования класса Trace.

Пример: Применение метода setTraceOn() и запись данных в протокол с помощью метода log

```
// Включение занесения диагностики, информации и предупреждений в протокол.
Trace.setTraceDiagnosticOn(true);
Trace.setTraceInformationOn(true);
Trace.setTraceWarningOn(true);
```

```
     // Включение трассировки.
     Trace.setTraceOn(true);

     // ... Запись информации в протокол.
     Trace.log(Trace.INFORMATION, "Вызов метода xxx класса xxx");

     // Отключение трассировки.
     Trace.setTraceOn(false);
```

Пример: Применение класса Trace

В следующем примере второй способ применения класса Trace является более предпочтительным.

```
     // Способ 1 - создание записи трассировки,
     // вызов метода log и определение с помощью класса трассировки
     // нужно ли записывать данные. Такой способ работает медленнее,
// чем приведенный ниже.
     String traceData = new String("Вход в класс xxx, данные = ");
     traceData = traceData + data + "состояние = " + state;
     Trace.log(Trace.INFORMATION, traceData);

     // Способ 2 - проверка состояния протокола перед созданием
     // записи. Этот метод эффективнее при отсутствии трассировки.
if (Trace.isTraceOn() && Trace.isTraceInformationOn())
     {
        String traceData = new String("Вход в класс xxx, данные = ");
        traceData = traceData + data + "состояние = " + state;
        Trace.log(Trace.INFORMATION, traceData);
     }
```

Пример: Трассировка отдельных компонентов

```
     // Создание строки с названием компонента. Создание объекта
// эффективнее, чем применение нескольких строковых литералов.
     String myComponent1 = "com.myCompany.xyzComponent";
     String myComponent2 = "com.myCompany.abcComponent";

     // Сохранение данных общей трассировки и трассировки отдельных компонентов в разных файлах.
     // Файл общей трассировки будет содержать всю информацию, а файл трассировки
     // отдельного компонента - только информацию, относящуюся к этому
     // компоненту. Если файл трассировки не указан, все данные трассировки
// передаются в стандартный вывод с указанием компонента
     // перед каждым сообщением.

     // Trace.setFileName("c:\\bit.bucket");
     // Trace.setFileName(myComponent1, "c:\\Component1.log");
     // Trace.setFileName(myComponent2, "c:\\Component2.log");

     Trace.setTraceOn(true);             // Включение трассировки.
     Trace.setTraceInformationOn(true);  // Запись информационных сообщений.

     // Занесение в протокол трассировки данных для отдельных
     // компонентов и общих данных трассировки.

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

Информация, связанная с данной

Trace Javadoc

User and group classes

The IBM Toolbox for Java user and group classes allow you to get a list of users and user groups on the server as well as information about each user through a Java program.

В частности, можно узнать дату и время последнего входа в систему, информацию о состоянии, дату изменения пароля, дату окончания срока действия пароля и класс пользователя. When you access the User

object, use the setSystem() method to set the system name and the setName() method to set the user name. After those steps, you use the loadUserInformation() method to get the information from the server.

The UserGroup object represents a special user whose user profile is a group profile. Using the getMembers() method, a list of users that are members of the group can be returned.

Программа на Java может перебрать все элементы множества по их номерам. All elements in the enumeration are User objects; for example:

```
// Создание объекта AS400.     AS400 system = new AS400 ("mySystem.myCompany.com");

// Создание объекта UserList.
UserList userList = new UserList (system);

// Получение списка пользователей и групп.
Enumeration enum = userList.getUsers ();

// Итерационная обработка списка.
while (enum.hasMoreElements ())
{
    User u = (User) enum.nextElement ();
    System.out.println  (u);
}
```

Получение информации о группах и пользователях

You use a UserList to get a list of the following:

- All users and groups
- Only groups
- All users who are members of groups
- All users who are not members of groups

The only property of the UserList object that must be set is the AS400 object that represents the system from which the list of users is to be retrieved.

По умолчанию выдается полный список пользователей. Use a combination of UserList methods setUserInfo() and setGroupInfo() to specify exactly which users are returned.

User Javadoc

UserGroup Javadoc

UserList Javadoc

AS400 Javadoc

“Пример: Применение класса UserList для получения списка пользователей указанной группы” на стр. 559
This source is an example of IBM Toolbox for Java UserList. This program lists all of the users in a given group.

Класс UserSpace

The UserSpace class represents a user space on the server. Required parameters are the name of the user space and the AS400 object that represents the server that has the user space.

С помощью методов этого класса можно выполнять следующие операции:

- Create a user space.
- Delete a user space.
- Read from a user space.
- Write to user space.
- Получать атрибуты пользовательских пространств. A Java program can get the initial value, length value, and automatic extendible attributes of a user space.

- Задавать атрибуты пользовательских пространств. A Java program can set the initial value, length value, and automatic extendible attributes of a user space.

Для работы с объектом UserSpace необходимо задать путь к программе в интегрированной файловой системе. Более подробная информация об этом приведена в разделе Пути в интегрированной файловой системе.

При работе с классом UserSpace объект AS400 устанавливает соединение с сервером. Информация об управлении соединениями приведена в разделе управление соединениями.

В примере ниже создается пользовательское пространство и в нем сохраняются данные.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
                // Создание объекта AS400.      AS400 sys = new AS400("mySystem.myCompany.com");

                // Создание объекта UserSpace
UserSpace US = new UserSpace(sys,
          "/QSYS.LIB/MYLIB.LIB/MYSPACE.USRSPC");

                // Создание пользовательского пространства
                // на сервере.
US.create(10240,                          // Начальный размер 10 Кб.
          true,                           // Заменить, если пространство существует
          " ",                            // Не расширять автоматически
          (byte) 0x00,                    // Начальное значение - пусто
          "Создано программой на Java",   // Описание пользовательского пространства
          "*USE");                        // Общие права доступа к пространству


                // Запись данных в пользовательское пространство с помощью метода write
US.write("Эта строка будет записана в пользовательское пространство.", 0);
```

UserSpace Javadoc

AS400 Javadoc

Классы Commtrace

Классы трассировки соединений IBM Toolbox for Java позволяют приложениям на Java работать с данными трассировки соединений описаний линий локальных сетей (Ethernet или Token-Ring). Пакет commtrace включает класс форматирования данных трассировки, который можно запустить в виде отдельной программы.

When you dump a communications trace for a server to a stream file, the information is saved in a binary format. Классы commtrace позволяют работать с различными компонентами потокового файла.

Примечание: Файлы трассировки соединений могут содержать конфиденциальную информацию, например, незашифрованные пароли. When the communications trace file is on server, only users who have *SERVICE special authority can access the trace data. Если файл находится в системе, убедитесь, что он надежно защищен. Дополнительная информация о трассировки соединений приведена в документах, ссылки на которые приведены в конце страницы.

С помощью классов commtrace можно выполнять следующие задачи:

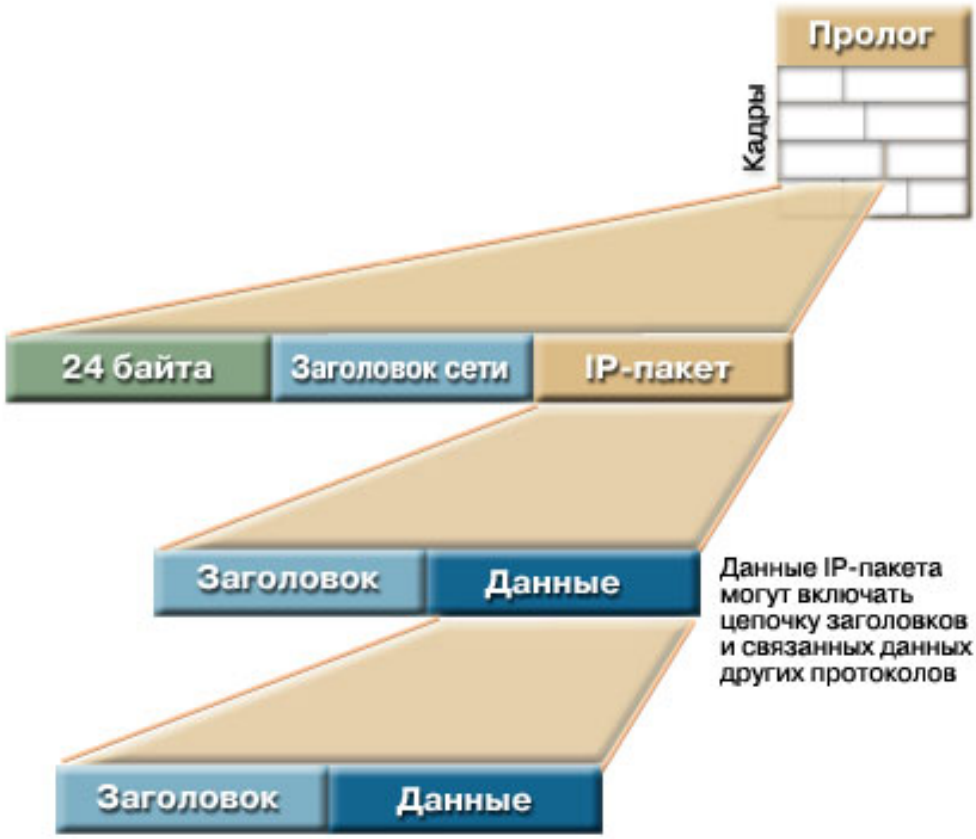
- Форматировать необработанные данные трассировки.
- Анализировать любые данные для поиска нужной информации. Анализировать можно как необработанные, так и отформатированные данные, если они были отформатированы с помощью классов commtrace.

Other classes in the com.ibm.as400.commtrace package not listed here are specific to the type of trace data that you want to work with. For more information about communications traces and about all the commtrace classes, see Communications trace.

Модель Commtrace

В приведенном ниже примере показано соответствие классов commtrace разделам файла трассировки соединений. На рисунке также показаны соглашения о присвоении имен, которые используются классами commtrace при работе с компонентами трассировки соединений.

Рисунок 1: Модель Commtrace



Each Frame in the trace file contains two initial sections (that provide general information about the contents of the frame) and the packet that the server transmitted between itself and a different point on the network.

В первом 24-разрядном разделе данных приведены общие сведения о содержимом фрейма, такие как его номер и длина данных. Для обработки этой информации служит класс Frame.

Классы Format и FormatProperties

The IBM Toolbox for Java Format class reads both raw data and formatted data from a communications trace. FormatProperties sets the properties for your Format object, such as start and end times, IP addresses, and ports.

The Format class serves as the interface between the calling program and the frames of the trace. The FormatProperties class enables you to set and retrieve properties that determine how the Format object behaves when it encounters information in the Frames of the trace.

Класс Format

С помощью класса Format можно получать необработанные данные трассировки и данные, которые были предварительно отформатированы с помощью классов commtrace.

Примечание: Классы commtrace не позволяют считывать данные трассировки, которые были отформатированы с помощью команды CL Печать трассировки соединений(PRTCMNTRC).

В этом случае необходимо с помощью класса Format выполнить анализ и отформатировать данные трассировки и сохранить их в файле или отправить на печать. Кроме того, можно создать графическое представление данных для просмотра с помощью отдельного приложения или в окне браузера. Для выбора каких-либо определенных данных можно передать их в программу на Java с помощью класса Format. Например, этот класс можно применять для получения IP-адресов из данных трассировки для последующего применения этих данных в программе.

Конструкторы Format поддерживают аргументы, представляющие необработанные данные, такие как объекты IFSFileInputStream, логические файлы и двоичные файлы трассировки. Для просмотра отформатированных данных трассировки необходимо воспользоваться конструктором Format по умолчанию, указав имя файла с помощью метода Format.openIFSFile() или Format.openLclFile().

Примеры

Ниже приведен пример просмотра сохраненной трассировки и форматирования двоичных данных трассировки.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Пример: Просмотр сохраненной трассировки

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

// Чтение пролога
System.out.println(fmt.getRecFromFile());
// Определение общего числа записей в TCP и других разделах трассировки
System.out.println("Total Records:" + fmt.getIntFromFile());
String rec;
// Чтение всех записей.
while((rec = fmt.getRecFromFile())!=null) {
System.out.println(rec);
```

Пример: Форматирование двоичного файла трассировки

```
// Создание объекта FormatProperties. По умолчанию выводятся все данные.
FormatProperties fmtprop = new FormatProperties();


Format fmt = new Format("/path/to/file");
// Указание свойств фильтрации для данного формата
fmt.setFilterProperties(fmtprop);
fmt.setOutFile("/path/to/output/file");
// Форматирование пролога
fmt.formatProlog();
// Форматирование трассировки и сохранение данных в указанном файле
fmt.toLclBinFile();
```

Запуск класса Format в качестве отдельной программы

Класс Format также можно запустить в виде отдельной программы. Дополнительная информация приведена в следующем разделе:

Запуск класса Format в качестве отдельной программы

Класс FormatProperties

Класс FormatProperties позволяет задавать и получать свойства объекта Format. Другими словами, при сохранении данных в файле с помощью класса Format класс FormatProperties выполняет фильтрацию сохраняемых данных.

Свойства определяют способ обработки данных фреймов трассировки соединений с помощью объекта Format. По умолчанию объект Format игнорирует свойства, для которых не заданы какие-либо значения.

Класс FormatProperties содержит константы для указания свойств. Свойства объекта Format применяются для фильтрации данных. Например, в приведенном ниже примере объект Format выведет окно диалога с индикатором выполнения и не выведет фреймы оповещения:

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

Большинство свойств объекта Format представлены в виде фильтров, позволяющих явно задавать нужные типы данных. После указания параметров фильтрации объект Format выводит только данные, соответствующие фильтрам. Например, в приведенном ниже примере будут показаны только фреймы, созданные в указанный промежуток времени:

```
FormatProperties prop = new FormatProperties();
  // Указание в фильтре начального и конечного времени: 22 июля 2002 г.,
  // 14:30 и 14:45 по Гринвичу.
  // Время задается в формате системного времени Unix(TM), для которого временем
  // отсчета является 1 января 1970 г. 00:00:00 по Гринвичу.
  prop.setStartTime("1027348200");
  prop.setEndTime("1027349100");
```

Пример

В приведенном ниже примере показано, как можно с помощью различных классов commtrace, включая Format и FormatProperties, выводить данные трассировки на экран:

“Пример: Применение классов Commtrace” на стр. 191

Format Javadoc

FormatProperties Javadoc

Запуск класса Format в качестве отдельной программы:

Класс Format можно как применять в программах на Java, так и запускать из командной строки в качестве отдельной программы для форматирования данных трассировки соединений. Программа подключает поток IFSFileOutputStream к заданному файлу вывода и сохраняет в нем данные.

Running format as a standalone utility enables you to format files by using the processing power and storage space of your server.

Запуск класса Format из командной строки

Для запуска класса Format из командной строки введите следующую команду:

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

, где список [опции] может содержать одну или несколько доступных опций. К ним относятся:

- Система, к которой необходимо подключиться
- ИД пользователя и пароль для этой системы

- Объект трассировки соединений, который необходимо проанализировать
- Файл, в котором следует сохранить результаты

For a complete list of available options, see the Javadoc reference documentation for the Format class.

Запуск класса Format из удаленной системы

Для запуска этого класса из удаленной системы воспользуйтесь классом JavaApplicationCall:

```
// Создание объекта JavaApplicationCall.
jaCall = new JavaApplicationCall(sys);
// Указание приложения на Java, которое необходимо запустить.
jaCall.setJavaApplication("com.ibm.as400.util.commtrace.Format");
// Указание значения переменной среды CLASSPATH,
// позволяющей JVM найти нужный класс.
jaCall.setClassPath("/QIBM/ProdData/OS400/JT400/lib/JT400Native.jar");

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

jaCall.setParameters(args2);

if (jaCall.run() != true) {
  // Сбой вызова
}
```

Информация, связанная с данной

Format Javadoc

Класс Prolog

The IBM Toolbox for Java Prolog class represents the initial 256-byte section of a communications trace for a LAN line description. Класс Prolog содержит общие сведения о трассировке, такие, как время начала и завершения, объем собранных данных и т.д. С помощью этого класса можно получать данные текущего раздела трассировки, которые можно напечатать, просмотреть, отфильтровать или обработать каким-нибудь другим образом.

Класс Prolog содержит методы, позволяющие выполнять различные задачи, в том числе:

- Получать значения полей объекта prolog, такие как описание трассировки, тип Ethernet, направление данных, IP-адрес и т.д.
- Возвращать отформатированные объекты String, содержащие все поля объекта prolog
- Тестировать данные полей объекта prolog

Пример

Ниже приведен пример того, как с помощью различных классов commtrace, включая Prolog, можно вывести на экран данные трассировки:

“Пример: Применение классов Commtrace” на стр. 191

Информация, связанная с данной

Prolog Javadoc

Класс Frame

Класс Frame представляет все данные, относящиеся к трассировке соединений описания линии LAN, в виде одной записи - фрейма.

Каждый объект Frame содержит три основных раздела данных, расположенных в следующем порядке:

1. Начальный 24-разрядный раздел, который содержит общую информацию о фрейме
2. Общую информацию о фрейме (представлена в виде класса LanHeader)

3. Данные пакета (представлены подклассом абстрактного класса IPacket)

Класс Frame служит для анализа и создания печатаемого представления данных фрейма. Он позволяет преобразовывать данные пакета в связанный список данных определенных форматов. For specific information about the possible formats for packet data in a frame and for general information about the structure of a frame, see “Модель Commtrace” на стр. 185.

С помощью методов класса Frame можно выполнять различные задачи, в том числе:

- Получать пакет данных
- Получать номер, состояние и тип фрейма
- Преобразовывать данные фрейма в отформатированные объекты String

Для обращения к данным пакета:

1. Получите пакет с помощью метода Frame.getPacket()
2. Получите данные заголовка с помощью метода Packet.getHeader()
3. После получения данных заголовка определите тип фрейма с помощью метода Header.getType()
4. Use the specific Header subclass to access the data associated with that header (the payload) and any additional headers

Пример

В приведенном ниже примере показано, как можно с помощью различных классов commtrace, включая Format и FormatProperties, выводить данные трассировки на экран:

“Пример: Применение классов Commtrace” на стр. 191

Класс LanHeader

The LanHeader class retrieves information from the section of data that occurs once, near the beginning of a frame, in the initial 24-byte section. This section typically contains hardware-specific information that includes general information about the frame, such as the frame number, and data length.

Класс LanHeader служит для анализа и вывода данных объекта LanHeader. Объект LanHeader содержит следующие данные:

- Разряд, указывающий на начало первого заголовка пакета
- MAC-адреса
- Адреса Token-Ring и данные маршрутизации

Класс LanHeader также содержит два метода, позволяющие возвращать отформатированные объекты String со следующими данными:

- Данные маршрутизации Token-Ring
- Исходные MAC-адреса, целевые MAC-адреса, формат и тип фрейма

Информация, связанная с данной

LanHeader Javadoc

Класс IPPacket

The IBM Toolbox for Java IPPacket class represents all the data packets that the network transmitted for this frame during the communications trace. IPPacket является абстрактным классом, поэтому заголовки и данные пакетов обрабатываются с помощью различных действительных подклассов.

Класс IPPacket содержит следующие подклассы:

- ARPPacket
- IP4Packet

- IP6Packet
- UnknownPacket

Классы Packet позволяют получать тип пакета и обращаться к необработанным данным (к заголовку и полезным данным), которые содержит пакет. Во всех подклассах применяются схожие конструкторы и содержится один дополнительный метод, который возвращает содержимое пакета в виде объекта String, который можно напечатать.

Всем конструкторам класса Packet в качестве аргумента передается массив байтов, но при применении класса ARPPacket необходимо также указывать целое число, задающее тип фрейма. При создании экземпляра класса Packet автоматически создается соответствующий объект Header.

С помощью методов классов Packet можно выполнять различные задачи, в том числе:
- Получать имя и тип пакета
- Задавать тип пакета
- Возвращать объект Header верхнего уровня, связанный с пакетом
- Возвращать все данные пакета в виде неотформатированного объекта String
- Возвращать часть данных пакета в виде отформатированного объекта String

  IPPacket Javadoc

  ARPPacket Javadoc

  IP4Packet Javadoc

  IP6Packet Javadoc

  UnknownPacket Javadoc

Класс Header

The Header class is the abstract superclass for creating classes that represent specific kinds of packet headers. В заголовки пакетов входят связанные данные (полезные данные), которые в свою очередь могут быть заголовками или полезными данными.

К числу подклассов класса Header относятся:
- ARPHeader
- ExtHeader
- ICMP4Header
- ICMP6Header
- IP4Header
- IP6Header
- TCPHeader
- UDPHeader
- UnknownHeader

Классы Header позволяют получать данные заголовка и полезные данные. Заголовок может содержать другие заголовки и их данные.

При создании экземпляра класса Packet автоматически создается соответствующий объект Header. С помощью методов классов Header можно выполнять различные задачи, в том числе:
- Возвращать длину, имя и тип заголовка
- Получать данные заголовка в виде массива байтов
- Получать следующий заголовок пакета
- Получать полезные данные в виде массива байтов, строки ASCII и шестнадцатеричной строки
- Возвращать все данные заголовка в виде неотформатированного объекта String

- Возвращать часть данных заголовка в виде отформатированного объекта String

  Header Javadoc

  ARPHeader Javadoc

  ExtHeader Javadoc

  ICMP4Header Javadoc

  ICMP6Header Javadoc

  IP4Header Javadoc

  IP6Header Javadoc

  TCPHeader Javadoc

  UDPHeader Javadoc

  UnknownHeader Javadoc

Пример: Применение классов Commtrace

This example uses the IBM Toolbox for Java commtrace classes to print communications trace data to a monitor by using a communications trace binary file as the source for the data.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////
//
// Пример применения классов commtrace для вывода на экран данных трассировки
// соединений из исходного двоичного файла трассировки.
//
//
// Формат вызова:
//    java CommTraceExample
//
////////////////////////////////////////////////////////////////////

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


public class CommTraceExample {


  public CommTraceExample() {
    // Создание объекта FormatProperties. По умолчанию выводятся все данные.
    FormatProperties fmtprop = new FormatProperties();

    Format fmt = new Format("/path/to/file");
    // Указание свойств фильтрации для данного формата
    fmt.setFilterProperties(fmtprop);
    fmt.formatProlog(); // Форматирование пролога

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

    // Если данные трассировки неправильны
    if (!pro.invalidData()) {
      Frame rec;

      // Получение записей
      while ((rec = fmt.getNextRecord()) != null) {

        // Вывод номера фрейма
        System.out.print("Запись:" + rec.getRecNum());
        // Вывод времени
        System.out.println(" Время:" + rec.getTime());
        // Получение данного пакета записей
        IPPacket p = rec.getPacket();
```

```
        // Получение первого заголовка
        Header h = p.getHeader();

        // Если применяется IPPacket IP6
        if (p.getType() == IPPacket.IP6) {

          // Если применяется Header IP6
          if (h.getType() == Header.IP6) {

            // Преобразование в IP6 для получения доступа к методам
            IP6Header ip6 = (IP6Header) h;

            System.out.println(h.getName() + " исх:" + ip6.getSrcAddr() + " цлв:" + ip6.getDstAddr());
            // Вывод заголовка в шестнадцатеричном формате
            System.out.println(ip6.printHexHeader());
            // Вывод заголовка в текстовом виде.
            System.out.println("Завершено " + h.getName() + ":\n" + ip6.toString(fmtprop));

            // Получение остальных заголовков
            while ((h = h.getNextHeader()) != null) {

              // При получении заголовка TCP
              if (h.getType() == Header.TCP) {
                // Преобразование данных для доступа к методам
                TCPHeader tcp = (TCPHeader) h;
                System.out.println(h.getName() +" исх:" + tcp.getSrcPort() +" цлв:" + tcp.getDstPort());
                System.out.println("Завершено " + h.getName() + ":\n" + tcp.toString(fmtprop));

                // При получении заголовка UDP
              } else if (h.getType() == Header.UDP) {
                // Преобразование данных для доступа к методам
                UDPHeader udp = (UDPHeader) h;
                System.out.println(h.getName() +" исх:" + udp.getSrcPort() + " цлв:" + udp.getDstPort());
                System.out.println("Завершено " + h.getName() + ":\n" + udp.toString(fmtprop));
              }
            }
          }
        }
      }
    }
  }

    public static void main(String[]args) {
        CommTraceExample e = new CommTraceExample();
    }
}
```

Классы HTML

The IBM Toolbox for Java HTML classes provide representations for many common HTML tag elements.

IBM Toolbox for Java HTML classes assist you in:

- Создавать формы и таблицы на страницах HTML
- Выравнивать текст
- Работать с различными тегами HTML
- Создавать исходные данные объекта форматирования (FO) расширяемого языка таблицы стилей (XSL)
- Изменять поддержку языка и направление ввода текста
- Создавать упорядоченные и неупорядоченные списки
- Создавать списки файлов и деревья HTML (а также их элементы)
- Добавлять атрибуты тегов, не определенные в классах HTML (например, атрибуты bgcolor и style)

The HTML classes implement the HTMLTagElement interface. Каждый класс создает тег HTML для элемента определенного типа. The tag may be retrieved using the getTag() method and can then be embedded into any HTML document. Теги, создаваемые классами, соответствуют спецификации HTML 3.2.

The HTML classes can work with servlet classes to get data from the server. Однако они могут применяться и отдельно, если данные для формы или таблицы создаются программой на Java.

Кроме того, класс HTMLDocument позволяет быстро создавать объекты HTML и исходные данные XSL FO. Данные XSL FO можно преобразовывать в документы PDF. Формат PDF обеспечивает электронное представление документов, полностью соответствующее печатным копиям.

Примечание: Файл jt400Servlet.jar содержит как класс HTML, так и класс Servlet. Для работы с этими классами из пакета com.ibm.as400.util.html необходимо добавить файл jt400Servlet.jar в переменную CLASSPATH.

Информация, связанная с данной

HTMLTagElement Javadoc

DirFilter Javadoc - Use to determine if a File object is a directory

HTMLFileFilter Javadoc - Use to determine if a File object is a file

URLEncoder Javadoc - Encodes delimiters to use in a URL string

URLParser Javadoc - Allows you to parse a URL string for the URI, properties, and reference

Класс BidiOrdering

The IBM Toolbox for Java BidiOrdering class represents an HTML tag that alters the language and direction of text. An HTML <BDO> string requires two attributes, one for language and the other for the direction of the text.

Класс BidiOrdering позволяет:

- Получать и задавать значение атрибута языка
- Получать и задавать направление ввода текста

For more information about using the <BDO> HTML tag, see the W3C Web site.

Пример: Применение класса BidiOrdering

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

В следующем примере создается объект BidiOrdering, после чего задается поддержка языка и направление ввода текста:

```
// Создание объекта BidiOrdering и выбор языка и направления ввода.
BidiOrdering bdo = new BidiOrdering();
bdo.setDirection(HTMLConstants.RTL);
bdo.setLanguage("AR");


// Формирование текста.
HTMLText text = new HTMLText("Текст на арабском.");
text.setBold(true);

// Добавление текста в объект BidiOrdering и получение тега HTML.
bdo.addItem(text);
bdo.getTag();
```

Последний оператор печати создаст следующий тег:

```
<bdo lang="AR" dir="rtl">
  <b>Some Arabic Text.</b>
</bdo>
```

When you use this tag in an HTML page, browsers that can understand the <BDO> tag display the example like this:

.txeT cibarA emoS

Информация, связанная с данной

BidiOrdering Javadoc

Класс HTMLAlign

The IBM Toolbox for Java HTMLAlign class enables you to align sections of your HTML document, instead of just aligning individual items such as paragraphs or headings.

The HTMLAlign class represents the <DIV> tag and its associated align attribute. Можно задать выравнивание по правому краю, по левому краю или по центру.

С помощью этого класса можно выполнять различные действия, в том числе:

- Add or remove items from the list of tags you want to align
- Get and set the alignment
- Get and set the direction of the text interpretation
- Get and set the language of the input element
- Get a string representation of the HTMLAlign object

Пример: Создание объектов HTMLAlign

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

В следующем примере формируется неупорядоченный список, а затем создается объект HTMLAlign, выравнивающий весь список:

```
// Создание неупорядоченного списка.
UnorderedList uList = new UnorderedList();
uList.setType(HTMLConstants.DISC);
UnorderedListItem uListItem1 = new UnorderedListItem();
uListItem1.setItemData(new HTMLText("Выровненный по центру неупорядоченный список"));
uList.addListItem(uListItem1);
UnorderedListItem uListItem2 = new UnorderedListItem();
uListItem2.setItemData(new HTMLText("Другой элемент"));
uList.addListItem(uListItem2);

// Выравнивание списка.
HTMLAlign align = new HTMLAlign(uList, HTMLConstants.CENTER);
System.out.println(align);
```

В данном примере создается следующий тег:

```
<div align="center">
<ul type="disc">
  <li>Centered unordered list</li>
  <li>Another item</li>
</ul>
```

На странице HTML этот тег будет выглядеть следующим образом:

- Выровненный по центру неупорядоченный список
- Другой элемент

Информация, связанная с данной

HTMLAlign Javadoc

Класс HTMLDocument

The HTMLDocument class enables you to use existing IBM Toolbox for Java HTML classes to create either HTML pages or Portable Document Format (PDF) documents.

При создании экземпляра HTMLDocument необходимо указывать, содержит ли он теги HTML или объектов форматирования (FO) XSL:

- При создании страниц HTML класс HTMLDocument предлагает простой способ группировки всех необходимых тегов HTML. Тем не менее, в некоторых случаях вид страниц HTML при печати отличается от вида в окне Web-браузера.
- При создании документов PDF класс HTMLDocument предлагает пользователю создать источник XSL FO, который содержит все данные, необходимые для создания документа PDF. Вид документов PDF при печати совпадает с их электронным представлением.

Для работы с классом HTMLDocument необходимо включить анализатор XML и обработчик XSLT в переменную среды CLASSPATH. Дополнительная информация приведена на следующих страницах:

- “Файлы Jar” на стр. 11
- “Анализатор XML и обработчик XSLT” на стр. 418

С полученными данными HTML и исходными данными XSL можно выполнять любые необходимые операции, например, открывать файлы HTML, сохранять XSL в файле или использовать потоковые данные в другой части программы на Java.

Дополнительная информация о создании страниц HTML и исходных данных XSL FO приведена на следующих страницах:

- “Создание данных HTML с помощью класса HTMLDocument”
- “Создание данных XSL FO с помощью класса HTMLDocument” на стр. 196
- “Примеры: Применение класса HTMLDocument” на стр. 198

  HTMLDocument Javadoc

Создание данных HTML с помощью класса HTMLDocument:

Класс HTMLDocument выполняет роль заменителя, который содержит информацию, необходимую для создания исходных данных объекта форматирования (FO) XSL или HTML. При создании страниц HTML класс HTMLDocument предлагает простой способ группировки всех необходимых тегов HTML.

Создание исходных данных HTML

При создании источника HTML класс HTMLDocument получает теги HTML из созданных пользователем объектов HTML. При этом все созданные элементы можно обработать одновременно с помощью метода HTMLDocument.getTag() или обработать каждый объект HTML по отдельности с помощью метода getTag().

HTMLDocument создает данные HTML, полностью соответствующие таким данным в программе Java, поэтому необходимо убедиться, что полученные документы HTML содержат правильную и полную информацию.

При вызове метода HTMLDocument.getTag() объект HTMLDocument выполняет следующие действия:

- Generates the opening <HTML> tag. At the end of the data, it generates the closing </HTML> tag.
- Преобразует объекты HTMLHead и HTMLMeta в теги HTML.
- Generates the opening <BODY> tag immediately after the <HEAD> tag. At the end of the data, just before the closing </HTML> tag, it generates the closing </BODY> tag.

  Примечание: If you do not specify a <HEAD> tag, HTMLDocument generates the <BODY> tag after the <HTML> tag.
- Преобразует остальные объекты HTML в теги HTML в соответствии с указаниями программы.

Примечание: HTMLDocument обрабатывает все теги HTML в соответствии с указаниями программы на Java, поэтому необходимо убедиться, что вызов тегов выполняется в правильном порядке.

Примеры: Применение класса HTMLDocument

Ниже приведен пример создания исходных данных HTML и источников XSL FO с помощью класса HTMLDocument:

“Пример: Создание источников HTML и XSL FO с помощью класса HTMLDocument.” на стр. 201

Справочная документация по Java

Дополнительная информация о классе HTMLDocument приведена в следующем разделе справочной документации по Java:

Класс HTMLDocument

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Создание данных XSL FO с помощью класса HTMLDocument:

Класс HTMLDocument выполняет роль заменителя, который содержит информацию, необходимую для создания исходных данных объекта форматирования (FO) XSL или HTML.

Созданный источник XSL FO соответствует модели форматирования XSL FO. В ней применяются прямоугольные элементы - объекты структуры, которые могут содержать отдельные элементы содержимого, такие как изображения, текст, другие объекты XSL FO, или не содержать ничего. Ниже перечислены четыре основных типа объектов структуры:

- Области, выполняющие роль контейнеров высшего уровня.
- Блоки, соответствующие элементам такого уровня, как абзацы или списки.
- Строки, соответствующие строкам или тексту внутри абзацев.
- Компоненты строк, которым соответствуют такие объекты, как отдельный символ, сноски или математические выражения.

Теги XSL FO, создаваемые IBM Toolbox for Java, соответствуют стандартам XSL, описанным в рекомендациях W3C. Дополнительная информация о XSL, источниках XSL FO и рекомендациях W3C приведена в разделе

Язык XSL версии 1.0

Создание исходных данных XSL FO

При создании источника XSL FO свойства класса HTMLDocument соответствуют тегам XSL FO, определяющим размер страницы, ее ориентацию и границы. Кроме того, HTMLDocument получает из многих классов HTML соответствующие теги XSL FO для элементов содержимого.

После создания источника XSL FO с помощью класса HTMLDocument можно с помощью программы форматирования XSL (например, класса XSLReportWriter) поместить элементы содержимого на страницы документа.

HTMLDocument создает два основных раздела исходных данных XSL FO:

- The first section contains the <fo:root> and <fo:layout-master-set> XSL FO tags that hold general page layout information for page height, page width, and page margins. Для того чтобы задать значения параметров разметки, с помощью соответствующих методов HTMLDocument задайте значения связанных свойств.
- The second section contains the XSL FO <fo:page-sequence> tag that holds the individual content elements. Для того чтобы задать такие элементы, которые являются классами HTML, необходимо получить соответствующий тег XSL FO из объекта HTML. Убедитесь, что в качестве элементов содержимого используются только классы HTML, в которых есть метод getFoTag().

  Примечание: При попытке получения тегов XSL FO из классов HTML, в которых нет метода getFoTag(), будет создан тег комментария.

Дополнительная информация о классах HTML, которые содержат методы для работы с тегами XSL FO, приведены в следующих разделах справочной документации по Java:

“Классы с поддержкой XSL FO” на стр. 198

После создания экземпляра HTMLDocument и задания свойств разметки необходимо получить теги XSL FO из объектов HTML с помощью методов setUseFO(), getFoTag() и getTag().

- При этом можно применять метод setUseFO() класса HTMLDocument или отдельных объектов HTML. При использовании метода setUseFO() можно получить теги XSL FO с помощью метода HTMLDocument.getTag().
- Кроме того, вы можете воспользоваться методом getFoTag() класса HTMLDocument или отдельных объектов HTML. Этот альтернативный способ позволяет создавать как источники XSL FO, так и источники HTML с помощью класса HTMLDocument или объектов HTML.

Пример: Применение класса HTMLDocument

После создания исходных данных XSL FO эти данные необходимо преобразовать в форму, доступную для просмотра и печати. Ниже приведены примеры создания исходных данных XSL FO (и источника HTML) и преобразования исходных данных XSL FO в документ PDF с помощью классов XSLReportWriter и Context:

“Пример: Создание источников HTML и XSL FO с помощью класса HTMLDocument.” на стр. 201

“Пример: Преобразование исходных данных XSL FO в документ PDF” на стр. 199

Справочная документация по Java

Дополнительная информация о классе HTMLDocument приведена в следующем разделе справочной документации по Java:

Класс HTMLDocument

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Классы с поддержкой XSL FO:

This topic describes the IBM Toolbox for Java classes that are compatible with the HTMLDocument class.

Многие классы HTML IBM Toolbox for Java содержат следующие методы, с помощью которых экземпляры этих классов могут работать с классом HTMLDocument:

- getFoTag()
- getTag()
- setUseFO()

Дополнительная информация о классе HTMLDocument и классах HTML, которые содержат методы для работы с XSL FO, приведены в следующих разделах справочной документации по Java:

- Класс HTMLDocument
- Класс BidiOrdering
- Класс HTMLAlign
- Класс HTMLHead
- Класс HTMLHeading
- HTMLImage
- Класс HTMLList
- Класс HTMLListItem
- HTMLTable
- Класс HTMLTableCaption
- Класс HTMLTableCell
- Класс HTMLTableHeader
- Класс HTMLTableRow
- Класс HTMLTagElement
- Класс OrderedList
- Класс UnorderedList

Примеры: Применение класса HTMLDocument:

В приведенных ниже примерах описаны способы создания исходных данных HTML и объектов форматирования (FO) XSL с помощью класса HTMLDocument.

Пример: Создание источников HTML и XSL FO с помощью класса HTMLDocument.

The following example shows how to generate both HTML source data and XSL FO source data at the same time:

“Пример: Создание источников HTML и XSL FO с помощью класса HTMLDocument.” на стр. 201

Пример: Преобразование исходных данных XSL FO в документ PDF

После создания исходных данных XSL FO эти данные необходимо преобразовать в форму, доступную для просмотра и печати. Ниже приведены примеры преобразования файла с исходными данными XSL FO в документ PDF с помощью классов XSLReportWriter и Context:

“Пример: Преобразование исходных данных XSL FO в документ PDF”

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Пример: Преобразование исходных данных XSL FO в документ PDF:

This example program should not be used as the XSLReportProcessor class is no longer supported.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример: Преобразование источника XSL FO в документ PDF.
//
// В этом примере с помощью классов ReportWriter IBM Toolbox for Java исходные данные
// XSL FO (созданные с помощью HTMLDocument) преобразуются в документ PDF.
//
// Для работы этого примера в переменной classpath должны быть указаны следующие файлы .jar.
//
//   composer.jar
//   outputwriters.jar
//   reportwriter.jar
//   x4j400.jar
//   xslparser.jar
//
// Они входят в состав IBM ToolBox for Java и расположены в каталоге
// /QIBM/ProdData/HTTP/Public/jt400/lib сервера.
//
// You will also need the class definition for
// org/apache/xerces/dom/NodeContainer, which resides
// in directory /QIBM/ProdData/OS400/xml/lib.
//
// Формат вызова:
//    ProcessXslFo имя-файла-FO имя-файла-PDF
//
////////////////////////////////////////////////////////////////////////////


import java.io.FileInputStream;
import java.io.FileOutputStream;

import java.awt.print.Paper;
import java.awt.print.PageFormat;
```

```
import org.w3c.dom.Document;

import com.ibm.xsl.composer.framework.Context;

import com.ibm.as400.util.reportwriter.pdfwriter.PDFContext;
import com.ibm.as400.util.reportwriter.processor.XSLReportProcessor;

public class ProcessXslFo
{
    public static void main( String args[] )
    {
        if (args.length != 2)
        {
            System.out.println("Usage:  java ProcessXslFo <fo file name> <pdf file name>");
            System.exit(0);
        }

        try
        {
            String inName = args[0];
            String outName = args[1];

            /* Ввод. Файл с XML FO. */
            FileInputStream fin = null;

            /* Вывод. В данном примере - файл PDF. */
            FileOutputStream fout = null;

            try
            {
                fin  = new FileInputStream(inName);
                fout = new FileOutputStream(outName);
            }
            catch (Exception e)
            {
                e.printStackTrace();
                System.exit(0);
            }

            /*
            * Формат страницы настройки.
            */
            Paper paper = new Paper();
            paper.setSize(612, 792);
            paper.setImageableArea(0, 0, 756, 936);

            PageFormat pageFormat = new PageFormat();
            pageFormat.setPaper(paper);

            /*
            * Создание контекста PDF. Указание имени файла вывода.
            */
            PDFContext pdfContext = new PDFContext(fout, pageFormat);

            /*
            * Создание экземпляра XSLReportProcessor.
            */
            XSLReportProcessor report = new XSLReportProcessor(pdfContext);

            /*
            * Открытие источника XML FO.
            */
            try
            {
                report.setXSLFOSource(fin);
            }
```

```
            catch (Exception e)
            {
                e.printStackTrace();
                System.exit(0);
            }

            /*
            * Обработка запроса.
            */
            try
            {
                report.processReport();
            }
            catch (Exception e)
            {
                e.printStackTrace();
                System.exit(0);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.exit(0);
        }

        /* выход */
        System.exit(0);
    }
}
```

Пример: Создание источников HTML и XSL FO с помощью класса HTMLDocument.:

This example uses the HTMLDocument class to generate HTML and XSL FO source data.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример: Создание с помощью класса HTMLDocument
// исходных данных HTML и XSL FO.
//
// В этой программе с помощью класса HTMLDocument
// создаются два файла: один из них содержит источник HTML,
// а второй - источник XSL FO.
//
// Формат вызова:
//    HTMLDocumentExample
//
////////////////////////////////////////////////////////////////////////////


import com.ibm.as400.util.html.*;
import java.*;
import java.io.*;
import java.lang.*;
import java.beans.PropertyVetoException;

public class HTMLDocumentExample
{
    public static void main(String[] args)
    {
        //Создание экземпляра HTMLDocument с необходимыми свойствами документов
        HTMLDocument doc = new HTMLDocument();

        //Указание свойств страницы и полей. Размеры задаются в дюймах.
```

```
doc.setPageWidth(8.5);
doc.setPageHeight(11);
doc.setMarginTop(1);
doc.setMarginBottom(1);
doc.setMarginLeft(1);
doc.setMarginRight(1);

//Создание заголовка страницы.
HTMLHead head = new HTMLHead();
//Указание названия заголовка
head.setTitle("Это заголовок страницы.");

//Создание нескольких заголовков
HTMLHeading h1 = new HTMLHeading(1, "Заголовок 1");
HTMLHeading h2 = new HTMLHeading(2, "Заголовок 2");
HTMLHeading h3 = new HTMLHeading(3, "Заголовок 3");
HTMLHeading h4 = new HTMLHeading(4, "Заголовок 4");
HTMLHeading h5 = new HTMLHeading(5, "Заголовок 5");
HTMLHeading h6 = new HTMLHeading(6, "Заголовок 6");

//Создание текста, который будет выводиться справа налево.
//Создание объекта BidiOrdering и установка направления
BidiOrdering bdo = new BidiOrdering();
bdo.setDirection(HTMLConstants.RTL);

//Создание текста
HTMLText text = new HTMLText("Это текст на арабском языке.");
//Добавление текста в объект двунаправленного языка
bdo.addItem(text);

// Создание UnorderedList.
UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
// Создание объектов UnorderedListItem и задание их содержимого.
UnorderedListItem listItem1 = new UnorderedListItem();
UnorderedListItem listItem2 = new UnorderedListItem();
listItem1.setItemData(new HTMLText("Первый элемент"));
listItem2.setItemData(new HTMLText("Второй элемент"));
// Добавление элементов списка в UnorderedList.
uList.addListItem(listItem1);
uList.addListItem(listItem2);

// Создание OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
// Создание объектов OrderedListItem.
OrderedListItem olistItem1 = new OrderedListItem();
OrderedListItem olistItem2 = new OrderedListItem();
OrderedListItem olistItem3 = new OrderedListItem();
// Задание содержимого объектов OrderedListItem.
olistItem1.setItemData(new HTMLText("Первый элемент"));
olistItem2.setItemData(new HTMLText("Второй элемент"));
olistItem3.setItemData(new HTMLText("Третий элемент"));
// Добавление элементов списка в OrderedList.
oList.addListItem(olistItem1);
oList.addListItem(olistItem2);
// Добавление (вложение) неупорядоченного списка в OrderedListItem2
oList.addList(uList);
// Добавление другого элемента в OrderedList
// после вложенного списка.
oList.addListItem(olistItem3);

// Создание объекта HTMLTable по умолчанию.
HTMLTable table = new HTMLTable();
try
{
    // Настройка атрибутов таблицы
    table.setAlignment(HTMLTable.LEFT);
    table.setBorderWidth(1);
```

```
            // Создание объекта HTMLTableCaption по умолчанию и содержимого таблицы.
            HTMLTableCaption caption = new HTMLTableCaption();
            caption.setElement("Баланс счетов заказчиков - 1 января 2000 года");

            // Добавление названия к таблице.
            table.setCaption(caption);

            // Создание заголовков столбцов и добавление их к таблице.
            HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("Счет"));
            HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("Имя"));
            HTMLTableHeader balance_header = new HTMLTableHeader(new HTMLText("Баланс"));

            table.addColumnHeader(account_header);
            table.addColumnHeader(name_header);
            table.addColumnHeader(balance_header);

            // Добавление строк к таблице. Каждой записи о заказчике соответствует одна строка таблицы.
int numCols = 3;
            for (int rowIndex=0; rowIndex< 5; rowIndex++)
            {
                HTMLTableRow row = new HTMLTableRow();
                row.setHorizontalAlignment(HTMLTableRow.CENTER);

                HTMLText account = new HTMLText("000" + rowIndex);
                HTMLText name = new HTMLText("Customer" + rowIndex);
                HTMLText balance = new HTMLText("" + (rowIndex + 1)*200);

                row.addColumn(new HTMLTableCell(account));
                row.addColumn(new HTMLTableCell(name));
                row.addColumn(new HTMLTableCell(balance));

                // Добавление строки к таблице.
                table.addRow(row);
            }
        }
        catch(Exception e)
        {
            System.out.println("Ошибка создания таблицы");
            System.exit(0);
        }

        //Добавление элементов в класс HTMLDocument
        doc.addElement(head);
        doc.addElement(h1);
        doc.addElement(h2);
        doc.addElement(h3);
        doc.addElement(h4);
        doc.addElement(h5);
        doc.addElement(h6);
        doc.addElement(oList);
        doc.addElement(table);
        doc.addElement(bdo);

        //Запись тегов fo в файл.
        try
        {
            FileOutputStream fout = new FileOutputStream("FOFILE.fo");
            PrintStream pout = new PrintStream(fout);
            pout.println(doc.getFOTag());
        }
        catch (Exception e)
        {
            System.out.println("Не удалось сохранить теги fo в файле FOFILE.fo");
        }

        //Запись тегов html в файл
```

```
        try
        {
            FileOutputStream htmlout = new FileOutputStream("HTMLFILE.html");
            PrintStream phtmlout = new PrintStream(htmlout);
            phtmlout.println(doc.getTag());
        }
        catch (Exception e)
        {
            System.out.println("Не удалось сохранить теги html в файле HTMLFILE.html");
        }

    }
}
```

Классы форм HTML

The IBM Toolbox for Java HTMLForm class represents an HTML form. Use these classes to make forms more easily than you could with CGI scripting.

Этот класс позволяет:

- Добавлять к форме такие элементы, как кнопки, гиперссылки и таблицы HTML
- Удалять элементы из формы
- Задавать прочие атрибуты формы, такие как метод отправки содержимого формы, список скрытых параметров и URL действия.

В конструкторе объекта HTMLForm указывается URL действия. Этот URL задает приложение сервера, которое будет обрабатывать данные формы. The action URL can be specified on the constructor or by setting the address using the setURL() method. Form attributes are set using various set methods and retrieved using various get methods.

Any HTML tag element may be added to an HTMLForm object using addElement() and removed using removeElement().

Of course, you can add other tag elements to a form, including HTMLText, HTMLHyperlink, and HTMLTable.

HTMLForm Javadoc

“Класс текста HTML” на стр. 220
The IBM Toolbox for Java HTMLText class allows you to access text properties for your HTML page. Using the HTMLText class, you can get, set, and check the status of many text attributes.

“Класс HTMLHyperlink” на стр. 213
The IBM Toolbox for Java HTMLHyperlink class represents an HTML hyperlink tag. С помощью этого класса вы можете добавить ссылку на страницу HTML.

“Классы таблиц HTML” на стр. 219
The IBM Toolbox for Java HTMLTable class allows you to easily set up tables that you can use in HTML pages.

“Пример: Применение классов форм HTML” на стр. 603
The following IBM Toolbox for Java example shows you how to use the HTML form classes.

“Пример вывода класса HTML” на стр. 613
These are some possible sample outputs you may get from running the HTML class example.

Классы FormInput:

The IBM Toolbox for Java FormInput class represents an input element in an HTML form.

The FormInput class allows you to:

- Get and set the name of an input element
- Get and set the size of an input element
- Get and set the initial value of an input element

Ниже приведен список классов, расширяющих класс FormInput. Эти классы позволяют создавать различные типы элементов ввода, получать и задавать различные атрибуты, а также получать теги HTML для элементов ввода:

- ButtonFormInput: кнопка
- FileFormInput: элемент выбора имени файла
- HiddenFormInput: скрытое поле ввода
- ImageFormInput: поле ввода изображения
- ResetFormInput: кнопка сброса
- SubmitFormInput: кнопка передачи данных формы
- TextFormInput: поле ввода одной строки текста с ограничением на максимальное число символов. Для ввода паролей этот класс расширен классом PasswordFormInput.
- ToggleFormInput: Represents a toggle input type for an HTML form. Программа может изменить или получить текст элемента и его состояние. Переключатель может быть одного из следующих типов:
  - RadioFormInput: Радиокнопка. Radio buttons may be placed in groups with the RadioFormInputGroup class; this creates a group of radio buttons where the user selects only one of the choices presented.
  - CheckboxFormInput: Переключатель, допускающий выбор нескольких элементов в группе. По умолчанию переключатель может быть включен или выключен.

  FormInput Javadoc

  ToggleFormInput Javadoc

  RadioFormInputGroup Javadoc

Класс ButtonFormInput:

The ButtonFormInput class represents a button element for an HTML form.

Ниже приведен пример создания объекта класса ButtonFormInput:

```
ButtonFormInput button = new ButtonFormInput(button1, Нажмите здесь, test());
System.out.println(button.getTag());
```

В результате этого примера будет создан следующий тег:

```
<input type="button" name="button1" value="Press Me" onclick="test()" />
```

Информация, связанная с данной

ButtonFormInput Javadoc

Класс FileFormInput:

The IBM Toolbox for Java FileFormInput class represents a file input type in an HTML form.

Ниже приведен пример создания объекта FileFormInput:

```
FileFormInput file = new FileFormInput("myFile");
System.out.println(file.getTag());
```

Этот текст создает следующий тег:

```
<input type="file" name="myFile" />
```

Информация, связанная с данной

FileFormInput Javadoc

Класс HiddenFormInput:

The IBM Toolbox for Java HiddenFormInput class represents a hidden input type in an HTML form.

Ниже приведен пример создания объекта HiddenFormInput:

```
HiddenFormInput hidden = new HiddenFormInput("account", "123456");
System.out.println(hidden.getTag()):
```

В следующем примере создается тег:

```
<input type="hidden" name="account" value="123456" />
```

Объект HiddenInputType не виден на странице HTML. Он применяется для того, чтобы передать информацию (в данном случае - номер счета) серверу.

Информация, связанная с данной

HiddenFormInput Javadoc

Класс ImageFormInput:

The ImageFormInput class represents an image input type in an HTML form.

Методы класса ImageFormInput позволяют управлять различными атрибутами этого элемента, в частности:

- Get or set the source
- Get or set the alignment
- Get or set the height
- Get or set the width

Пример: Создание объекта ImageFormInput

Ниже приведен пример создания объекта класса ImageFormInput:

```
ImageFormInput image = new ImageFormInput("myPicture", "myPicture.gif");
image.setAlignment(HTMLConstants.TOP);
image.setHeight(81);
image.setWidth(100);
```

Приведенный выше код создаст следующий тег:

```
<input type="image" name="MyPicture" src="myPicture.gif" align="top" height="81" width="100" />
```

Информация, связанная с данной

ImageFormInput Javadoc

Класс ResetFormInput:

The ResetFormInput class represents a reset button input type in an HTML form.

Ниже приведен пример создания объекта ResetFormInput:

```
ResetFormInput reset = new ResetFormInput();
reset.setValue("Сброс");
System.out.println(reset.getTag());
```

Приведенный выше фрагмент программы создает следующий код HTML:

```
<input type="reset" value="Reset" />
```

Информация, связанная с данной

ResetFormInput Javadoc

Класс SubmitFormInput:

The SubmitFormInput class represents a submit button input type in an HTML form.

Ниже приведен пример кода, в котором создается объект SubmitFormInput:

```
SubmitFormInput submit = new SubmitFormInput();
submit.setValue("Отправить");
System.out.println(submit.getTag());
```

После обработки этого примера кода будет получен следующий вывод:

```
<input type="submit" value="Send" />
```

Информация, связанная с данной

SubmitFormInput Javadoc

Класс TextFormInput:

The TextFormInput class represents a single line text input type in an HTML form. The TextFormInput class provides methods that let you get and set the maximum number of characters a user can enter in the text field.

Следующий пример иллюстрирует создание объекта TextFormInput:

```
TextFormInput text = new TextFormInput(ИД-пользователя);
text.setSize(40);
System.out.println(text.getTag());
```

Приведенный выше фрагмент программы создает следующий тег:

```
<input type="text" name="userID" size="40" />
```

Информация, связанная с данной

TextFormInput Javadoc

Класс PasswordFormInput:

The PasswordFormInput class represents a password input field type in an HTML form.

Ниже приведен пример создания нового объекта класса PasswordFormInput:

```
PasswordFormInput pwd = new PasswordFormInput(password);
pwd.setSize(12);
System.out.println(pwd.getTag());
```

Приведенный выше фрагмент программы создает следующий тег:

```
<input type="password" name="password" size="12" />
```

Информация, связанная с данной

PasswordFormInput Javadoc

Класс RadioFormInput:

The RadioFormInput class represents a radio button input type in an HTML form. При создании радиокнопку можно инициализировать как выбранную.

Несколько радиокнопок с одинаковым именем управляющего элемента образуют группу. The RadioFormInputGroup class creates radio button groups. В группе можно выбрать не более одной кнопки одновременно. При создании группы определенную кнопку можно инициализировать как выбранную.

Ниже приведен пример фрагмента программы, создающего объект RadioFormInput:

```
RadioFormInput radio = new RadioFormInput("age", "twentysomething", "Возраст 20-29", true);
System.out.println(radio.getTag());
```

Приведенный выше код создаст следующий тег:

```
<input type="radio" name="age" value="twentysomething" checked="checked" />
```

RadioFormInput Javadoc

RadioFormInputGroup Javadoc

Класс CheckboxFormInput:

The IBM Toolbox for Java CheckboxFormInput class represents a checkbox input type in an HTML form. Переключатели позволяют выбрать несколько вариантов одновременно.

Ниже приведен пример создания объекта класса CheckboxFormInput:

```
CheckboxFormInput checkbox = new CheckboxFormInput(uscitizen, yes, textLabel, true);
System.out.println(checkbox.getTag());
```

Приведенный выше код создаст следующий тег:

```
<input type="checkbox" name="uscitizen" value="yes" checked="checked" /> textLabel
```

Класс LayoutFormPanel:

The IBM Toolbox for Java LayoutFormPanel class represents a layout of form elements for an HTML form. Вы можете воспользоваться методами LayoutFormPanel для добавления и удаления элементов панели и получения общего числа элементов формы.

Существует два варианта размещения элементов:

- GridLayoutFormPanel: Represents a grid layout of form elements for an HTML form
- LineLayoutFormPanel: Represents a line layout of form elements for an HTML form

LayoutFormPanel Javadoc

“GridLayoutFormPanel”
The GridLayoutFormPanel class represents a grid layout of form elements. Вы можете использовать эту разметку для формы HTML, если требуется организовать вывод элементов с фиксированным числом колонок.

“Класс LineLayoutFormPanel” на стр. 209
The LineLayoutFormPanel class represents a line layout of form elements for an HTML form. Все элементы формы расположены на панели в один ряд.

GridLayoutFormPanel:

The GridLayoutFormPanel class represents a grid layout of form elements. Вы можете использовать эту разметку для формы HTML, если требуется организовать вывод элементов с фиксированным числом колонок.

В следующем примере создается объект GridLayoutFormPanel с двумя колонками:

```
// Создание поля текстового ввода для имени системы.
LabelFormElement sysPrompt = new LabelFormElement("Система:");
TextFormInput system = new TextFormInput("System");

// Создание поля текстового ввода для ИД пользователя.
LabelFormElement userPrompt = new LabelFormElement("Пользователь:");
TextFormInput user = new TextFormInput("User");

// Создание поля ввода пароля для пароля.
LabelFormElement passwordPrompt = new LabelFormElement("Пароль:");
PasswordFormInput password = new PasswordFormInput("Password");

// Создание объекта GridLayoutFormPanel с двумя колонками и добавление в него элементов формы.
GridLayoutFormPanel panel = new GridLayoutFormPanel(2);
panel.addElement(sysPrompt);
panel.addElement(system);
panel.addElement(userPrompt);
panel.addElement(user);
panel.addElement(passwordPrompt);
panel.addElement(password);
```

```
// Создание кнопки обработки формы.
SubmitFormInput logonButton = new SubmitFormInput("logon", "Войти в систему");

// Создание объекта HTMLForm и добавление в него панели.
HTMLForm form = new HTMLForm(servletURI);
form.addElement(panel);
form.addElement(logonButton);
```

В результате этого примера будет создан следующий код HTML:

```
<form action=servletURI method="get">
<table border="0">
<tr>
<td>System:</td>
<td><input type="text" name="System" /></td>
</tr>
<tr>
<td>User:</td>
<td><input type="text" name="User" /></td>
</tr>
<tr>
<td>Password:</td>
<td><input type="password" name="Password" /></td>
</tr>
</table>
<input type="submit" name="logon" value="Logon" />
</form>
```

Информация, связанная с данной

GridLayoutFormPanel Javadoc

Класс LineLayoutFormPanel:

The LineLayoutFormPanel class represents a line layout of form elements for an HTML form. Все элементы формы расположены на панели в один ряд.

Пример: Применение объекта LineLayoutFormPanel

В приведенном ниже примере создается объект LineLayoutFormPanel и добавляются два элемента формы.

```
CheckboxFormInput privacyCheckbox =
   new CheckboxFormInput("confidential", "yes", "Confidential", true);
CheckboxFormInput mailCheckbox =
   new CheckboxFormInput("mailingList", "yes", "Join our mailing list", false);
LineLayoutFormPanel panel = new LineLayoutFormPanel();
panel.addElement(privacyCheckbox);
panel.addElement(mailCheckbox);
String tag = panel.getTag();
```

Приведенный выше фрагмент программы создает следующий код HTML:

```
<input type="checkbox" name="confidential" value="yes" checked="checked" /> Confidential
<input type="checkbox" name="mailingList" value="yes" /> Join our mailing list <br/>
```

Информация, связанная с данной

LineLayoutFormPanel Javadoc

Класс TextAreaFormElement:

The TextAreaFormElement class represents a text area element in an HTML form. You specify the size of the text area by setting the number of rows and columns. You can determine the size that a text area element is set for with the getRows() and getColumns() methods.

You set the initial text within the text area with the setText() method. You use the getText() method to see what the initial text has been set to.

Следующий пример иллюстрирует создание класса TextAreaFormElement:

```
TextAreaFormElement textArea = new TextAreaFormElement("foo", 3, 40);
textArea.setText("Значение TEXTAREA по умолчанию");
System.out.println(textArea.getTag());
```

Приведенный выше фрагмент программы создает следующий код HTML:

```
<form>
<textarea name="foo" rows="3" cols="40">
Значение TEXTAREA по умолчанию
</textarea>
</form>
```

TextAreaFormElement Javadoc

Класс LabelFormElement:

The LabelFormElement class represents a label for an HTML form element.

Он позволяет создать метку для таких элементов формы HTML, как область текста или поле для ввода пароля. The label is one line of text that you set using the setLabel() method. Этот текст не рассматривается как ввод пользователя. Метка просто поясняет назначение элемента формы.

Пример: Применение класса LabelFormElement

Ниже приведен пример создания объекта класса LabelFormElement:

```
LabelFormElement label = new LabelFormElement("Баланс");
System.out.println(label.getTag());
```

В результате будет получена следующая метка:

```
Баланс
```

Информация, связанная с данной

LabelFormElement Javadoc

Класс SelectFormElement:

The SelectFormElement class represents a select input type for an HTML form. You can add and remove various options within the select element.

В классе SelectFormElement предусмотрены методы для просмотра и изменения атрибутов поля со списком:

- Use setMultiple() to set whether or not the user can select more than one option
- Use getOptionCount() to determine how many elements are in the option layout
- Use setSize() to set the number of options visible within the select element and use getSize() to determine the number of visible options.

В приведенном ниже примере создается объект SelectFormElement с тремя элементами. Объект SelectFormElement с именем *list* выделен. При добавлении первых двух элементов указывается текст, имя элемента и атрибут выделения. В качестве третьего элемента добавляется объект SelectOption.

```
SelectFormElement list = new SelectFormElement("list1");
SelectOption option1 = list.addOption("Вариант1", "opt1");
SelectOption option2 = list.addOption("Вариант2", "opt2", false);
SelectOption option3 = new SelectOption("Вариант3", "opt3", true);
list.addOption(option3);
System.out.println(list.getTag());
```

Приведенный выше пример кода преобразуется в следующий код HTML:

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

Ссылки, связанные с данной

"Класс SelectOption"
The SelectOption class represents an option in an HTML SelectFormElement. You use the option form element in a select form.

Информация, связанная с данной

SelectFormElement Javadoc

Класс SelectOption:

The SelectOption class represents an option in an HTML SelectFormElement. You use the option form element in a select form.

В этом классе предусмотрены методы для получения и изменения атрибутов SelectOption. For instance, you can set whether the option defaults to being selected. You can also set the input value it will use when the form is submitted.

В следующем примере создается поле со списком, содержащее три объекта SelectOption. Все объекты SelectOption выделены. Они называются *Вариант1, Вариант2* и *Вариант3*. По умолчанию выбран объект *Вариант3*.

```
SelectFormElement list = new SelectFormElement("list1");
SelectOption option1 = list.addOption("Вариант1", "opt1");
SelectOption option2 = list.addOption("Вариант2", "opt2", false);
SelectOption option3 = new SelectOption("Вариант3", "opt3", true);
list.addOption(option3);
System.out.println(list.getTag());
```

Приведенный выше пример кода соответствует следующему тегу HTML:

```
<select name="list1">
<option value="opt1">Option1</option>
<option value="opt2">Option2</option>
<option value="opt3" selected="selected">Option3</option>
</select>
```

Ссылки, связанные с данной

"Класс SelectFormElement" на стр. 210
The SelectFormElement class represents a select input type for an HTML form. You can add and remove various options within the select element.

Информация, связанная с данной

SelectOption Javadoc

Класс RadioFormInputGroup:

The RadioFormInputGroup class represents a group of RadioFormInput objects. Пользователь может выбирать из RadioFormInputGroup только по одному объекту RadioFormInput.

Методы класса RadioFormInputGroup позволяют работать с различными атрибутами группы радиокнопок. С помощью этих методов можно выполнять следующие операции:

- Add a radio button
- Remove a radio button
- Get or set the name of the radio group

В следующем примере создается группа радиокнопок:

```
// Создание нескольких радиокнопок.
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);
// Создание группы радиокнопок и добавление в нее радиокнопок.
RadioFormInputGroup ageGroup = new RadioFormInputGroup("age");
ageGroup.add(radio0);
ageGroup.add(radio1);
ageGroup.add(radio2);
ageGroup.add(radio3);
System.out.println(ageGroup.getTag());
```

Приведенный выше фрагмент программы создает следующий код 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
```

Информация, связанная с данной

RadioFormInputGroup Javadoc

RadioFormInput Javadoc

Класс HTMLHead

The IBM Toolbox for Java HTMLHead class represents an HTML head tag. Раздел заголовка страницы HTML содержит открывающий и закрывающий теги заголовка, которые, как правило, содержат другие теги. Чаще всего тег заголовка содержит тег названия и некоторые мета-теги.

Конструкторы класса HTMLHead позволяют создавать пустые теги заголовка, теги, содержащие теги названия, и теги, содержащие теги названия и мета-теги. В пустой объект HTMLHead можно быстро добавить тег названия и мета-теги.

Методы класса HTMLHead позволяют задавать и получать теги названия и мета-теги. Определить содержимое мета-тегов можно с помощью класса HTMLMeta.

В следующем примере показан один из способов создания объекта HTMLHead:

```
// Создание пустого объекта HTMLHead.
HTMLHead head = new HTMLHead("Моя главная страница");

// Добавление названия.
head.setTitle("Моя главная страница");

// Указание мета-данных и добавление их в экземпляр HTMLHead.
HTMLMeta meta = new HTMLMeta("Content-Type", "text/html; charset=iso-8859-1");
head.addMetaInformation(meta);
```

Ниже приведен вывод примера программы для тега HTMLHead:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>My main page</title>
</head>
```

Ссылки, связанные с данной

“Класс HTMLMeta” на стр. 217
The IBM Toolbox for Java HTMLMeta class represents meta-information used within an HTMLHead tag. Атрибуты тегов META используются при идентификации, индексации и определении информации в документе HTML.

Информация, связанная с данной

HTMLHead Javadoc

Класс HTMLHeading

The IBM Toolbox for Java HTMLHeading class represents an HTML heading. Для заголовка можно задать атрибут выравнивания и уровень от 1 (максимальный шрифт, наибольшая важность) до 6.

Класс HTMLHeading содержит методы, позволяющие:

- Get and set the text for the heading
- Get and set the level of the heading
- Get and set the alignment of the heading
- Get and set the direction of the text interpretation
- Get and set the language of the input element
- Get a String representation of the HTMLHeading object

Пример: Создание объектов HTMLHeading

В следующем примере создаются три объекта HTMLHeading:

```
// Создание и вывод трех объектов HTMLHeading.
HTMLHeading h1 = new HTMLHeading(1, "Заголовок", HTMLConstants.LEFT);
HTMLHeading h2 = new HTMLHeading(2, "Подзаголовок", HTMLConstants.CENTER);
HTMLHeading h3 = new HTMLHeading(3, "Элемент", HTMLConstants.RIGHT);
System.out.print(h1 + "\r\n" + h2 + "\r\n" + h3);
```

В предыдущем примере создаются следующие теги:

```
<h1 align="left">Heading</h1>
<h2 align="center">Subheading</h2>
<h3 align="right">Item</h3>
```

Информация, связанная с данной

HTMLHeading Javadoc

Класс HTMLHyperlink

The IBM Toolbox for Java HTMLHyperlink class represents an HTML hyperlink tag. С помощью этого класса вы можете добавить ссылку на страницу HTML.

Кроме того, этот класс позволяет получать и задавать следующие атрибуты ссылки:

- Get and set the Uniform Resource Identifier for the link
- Get or set the title for the link
- Get or set the target frame for the link

Класс HTMLHyperlink позволяет получить полную гиперссылку с заданными атрибутами для вставки в документ HTML.

Ниже приведен пример применения класса HTMLHyperlink:

```
// Создание гиперссылки на домашнюю страницу IBM Toolbox for Java.
HTMLHyperlink toolbox = new HTMLHyperlink("http://www.ibm.com/as400/toolbox",
    "Домашняя страница IBM Toolbox for Java");
toolbox.setTarget(TARGET_BLANK);

// Вывод тега.
System.out.println(toolbox.toString());
```

В результате выполнения приведенного кода будет создан следующий тег:

```
<a href="http://www.ibm.com/as400/toolbox">IBM Toolbox for Java home page</a>
```

Информация, связанная с данной

HTMLHyperlink Javadoc

Класс HTMLImage

The HTMLImage class allows you to create image tags for your HTML page.

С помощью методов класса HTMLImage можно получать и задавать различные атрибуты изображения, в частности:

- Get or set the height of the image
- Get or set the width of the image
- Get or set the name of the image
- Get or set the alternate text for the image
- Get or set the horizontal space around the image
- Get or set the vertical space around the image
- Get or set the absolute or relative reference to the image
- Retrieve a string representation of the HTMLImage object

В следующем примере показан один из способов создания объекта HTMLImage:

```
// Создание объекта HTMLImage.
HTMLImage image = new HTMLImage("http://myWebSite/picture.gif", "Альтернативный текст");
image.setHeight(94);
image.setWidth(105);
System.out.println(image);
```

Последний оператор печати создаст следующий тег на одной строке. Перенос строк применяется только для простоты восприятия.

```
<img src="http://myWebSite/picture.gif"
     alt="Альтернативный текст"
     height="94" width="105" />
```

Информация, связанная с данной

HTMLImage Javadoc

Классы HTMLList

The IBM Toolbox for Java HTMLList classes allow you to easily create lists within your HTML pages. Эти классы позволяют задавать атрибуты списков и их элементов.

In particular, the parent class HTMLList provides a method to produce a compact list that displays items in as small a vertical space as possible.

- Methods for HTMLList include:
  - Compact the list
  - Add and remove items from the list
  - Add and remove lists from the list (making it possible to nest lists)
- Класс HTMLListItem содержит следующие методы:
  - Get and set the contents of the item
  - Get and set the direction of the text interpretation
  - Get and set the language of the input element

Для создания списков HTML воспользуйтесь следующими подклассами классов HTMLList и HTMLListItem:

- OrderedList и OrderedListItem
- UnorderedList и UnorderedListItem

Ознакомьтесь со следующими примерами фрагментов кода:

- Пример: Создание упорядоченных списков
- Пример: Создание неупорядоченных списков

- Пример: Создание вложенных списков

Классы OrderedList и OrderedListItem

Классы OrderedList иOrderedListItem служат для создания упорядоченных списков на страницах HTML.

- Класс OrderedList содержит следующие методы:
  - Get and set the starting number for the first item in the list
  - Get and set the type (or style) for the item numbers
- Класс OrderedListItem содержит следующие методы:
  - Get and set the number for the item
  - Get and set the type (or style) for the item number

С помощью методов класса OrderedListItem можно переопределять номер и тип конкретного элемента списка.

Ознакомьтесь с примером создания упорядоченных списков.

Классы UnorderedList и UnorderedListItem

Классы UnorderedList и UnorderedListItem служат для создания неупорядоченных списков на страницах HTML.

- Класс UnorderedList содержит следующие методы:
  - Get and set the type (or style) for the items
- Класс UnorderedListItem содержит следующие методы:
  - Get and set the type (or style) for the item

Ознакомьтесь с примером создания неупорядоченных списков.

Пример: Применение классов HTMLList

Ниже приведены примеры использования классов HTMLList для создания упорядоченных, неупорядоченных и вложенных списков.

Пример: Создание упорядоченных списков

В следующем примере создается упорядоченный список:

```
     // Создание OrderedList.
OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
     // Создание объектов OrderedListItem.
OrderedListItem listItem1 = new OrderedListItem();
OrderedListItem listItem2 = new OrderedListItem();
     // Задание содержимого объектов OrderedListItem.
listItem1.setItemData(new HTMLText("Первый элемент"));
listItem2.setItemData(new HTMLText("Второй элемент"));
     // Добавление элементов списка в OrderedList.
oList.addListItem(listItem1);
oList.addListItem(listItem2);
System.out.println(oList.getTag());
```

В предыдущем примере создаются следующие теги:

```
<ol type="i">
<li>First item</li>
<li>Second item</li>
</ol>
```

Пример: Создание неупорядоченных списков

В следующем примере создается неупорядоченный список:

```
        // Создание UnorderedList.
    UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
        // Создание объектов UnorderedListItem.
    UnorderedListItem listItem1 = new UnorderedListItem();
    UnorderedListItem listItem2 = new UnorderedListItem();
        // Задание содержимого объектов UnorderedListItem.
    listItem1.setItemData(new HTMLText("Первый элемент"));
    listItem2.setItemData(new HTMLText("Второй элемент"));
        // Добавление элементов списка в UnorderedList.
    uList.addListItem(listItem1);
    uList.addListItem(listItem2);
    System.out.println(uList.getTag());
```

В предыдущем примере создаются следующие теги:

```
<ul type="square">
<li>First item</li>
<li>Second item</li>
</ul>
```

Пример: Создание вложенных списков

В следующем примере создается вложенный список:

```
        // Создание UnorderedList.
    UnorderedList uList = new UnorderedList(HTMLConstants.SQUARE);
        // Создание объектов UnorderedListItem и задание их содержимого.
    UnorderedListItem listItem1 = new UnorderedListItem();
    UnorderedListItem listItem2 = new UnorderedListItem();
    listItem1.setItemData(new HTMLText("Первый элемент"));
    listItem2.setItemData(new HTMLText("Второй элемент"));
        // Добавление элементов списка в UnorderedList.
    uList.addListItem(listItem1);
    uList.addListItem(listItem2);

        // Создание OrderedList.
    OrderedList oList = new OrderedList(HTMLConstants.SMALL_ROMAN);
        // Создание объектов OrderedListItem.
    OrderedListItem listItem1 = new OrderedListItem();
    OrderedListItem listItem2 = new OrderedListItem();
    OrderedListItem listItem3 = new OrderedListItem();
        // Задание содержимого объектов OrderedListItem.
    listItem1.setItemData(new HTMLText("Первый элемент"));
    listItem2.setItemData(new HTMLText("Второй элемент"));
    listItem3.setItemData(new HTMLText("Третий элемент"));
        // Добавление элементов списка в OrderedList.
    oList.addListItem(listItem1);
    oList.addListItem(listItem2);
        // Добавление (вложение) неупорядоченного списка в OrderedListItem2
    oList.addList(uList);
        // Добавление другого элемента в OrderedList
        // после вложенного списка.
    oList.addListItem(listItem3);
    System.out.println(oList.getTag());
```

В предыдущем примере создаются следующие теги:

```
    <ol type="i">
    <li>First item</li>
    <li>Second item</li>
    <ul type="square">
    <li>First item</li>
```

```
<li>Second item</li>
</ul>
<li>Third item</li>
</ol>
```

HTMLList Javadoc

HTMLListItem Javadoc

Класс HTMLMeta

The IBM Toolbox for Java HTMLMeta class represents meta-information used within an HTMLHead tag. Атрибуты тегов META используются при идентификации, индексации и определении информации в документе HTML.

В теге META задаются следующие атрибуты:

- NAME - имя, связанное с содержимым тега META
- CONTENT - значения, связанные с атрибутом NAME
- HTTP-EQUIV - информация, собранная серверами HTTP, для заголовков ответных сообщений
- LANG - язык
- URL - адрес страницы, на которую перенаправляется пользователь с текущей страницы

Например, для того чтобы упростить поисковым серверам сбор информации о содержимом страницы, можно задать следующий тег META:

```
<META name="keywords" lang="en-us" content="games, cards, bridge">
```

Кроме того, тег HTMLMeta применяется для перенаправления пользователей с одной страницы на другую.

Класс HTMLMeta содержит следующие методы:

- Get and set the NAME attribute
- Get and set the CONTENT attribute
- Get and set the HTTP-EQUIV attribute
- Get and set the LANG attribute
- Get and set the URL attribute

Пример: Создание тегов META

В следующем примере создается два тега META:

```
// Создание тега META для поисковых серверов.
HTMLMeta meta1 = new HTMLMeta();
meta1.setName("keywords");
meta1.setLang("en-us");
meta1.setContent("games, cards, bridge");
// Создание тега META, используемого кэшем для обновления страницы.
HTMLMeta meta2 = new HTMLMeta("Expires", "Mon, 01 Jun 2000 12:00:00 GMT");
System.out.print(meta1 + "\r\n" + meta2);
```

В предыдущем примере создаются следующие теги:

```
<meta name="keywords" content="games, cards, bridge">
<meta http-equiv="Expires" content="Mon, 01 Jun 2000 12:00:00 GMT">
```

Информация, связанная с данной

HTMLMeta Javadoc

Класс HTMLParameter

The HTMLParameter class represents the parameters you can use with the HTMLServlet class. У каждого параметра есть имя и значение.

Методы класса HTMLParameter позволяют:

- Get and set the name of the parameter
- Get and set the value of the parameter

Пример: Создание тегов HTMLParameter

В следующем примере создается тег HTMLParameter:

```
      // Создать объект HTMLServletParameter.
HTMLParameter parm = new HTMLParameter ("age", "21");
System.out.println(parm);
```

В данном примере создается следующий тег:

```
<param name="age" value="21">
```

Информация, связанная с данной

HTMLParameter Javadoc

Класс HTMLServlet

The HTMLServlet class represents a server-side include. В объекте сервлета задается имя сервлета и, при необходимости, его расположение. По умолчанию применяется расположение в локальной системе.

The HTMLServlet class works with the HTMLParameter class, which specifies the parameters available to the servlet.

Класс HTMLServlet содержит следующие методы:

- Add and remove HTMLParameters from the servlet tag
- Get and set the location of the servlet
- Get and set the name of the servlet
- Get and set the alternate text of the servlet

Пример: Создание тегов HTMLServlet

В следующем примере создается тег HTMLServlet:

```
// Создание HTMLServlet.
HTMLServlet servlet = new HTMLServlet("myServlet", "http://server:port/dir");

// Создание параметра и добавление его в сервлет.
HTMLParameter param = new HTMLParameter("parm1", "value1");
servlet.addParameter(param);

// Создание и добавление второго параметра
HTMLParameter param2 = servlet.add("parm2", "value2");

// Альтернативный текст на случай, если Web-сервер не поддерживает тег сервлета.
servlet.setText("Web-сервер, предоставивший эту страницу, не поддерживает тег SERVLET.");
System.out.println(servlet);
```

В предыдущем примере создаются следующие теги:

```
<servlet name="myServlet" codebase="http://server:port/dir">
<param name="parm1" value="value1">
<param name="parm2" value="value2">
Web-сервер, предоставивший эту страницу, не поддерживает тег SERVLET.
</servlet>
```

HTMLServlet Javadoc

“Класс HTMLParameter” на стр. 217
The HTMLParameter class represents the parameters you can use with the HTMLServlet class. У каждого параметра есть имя и значение.

Классы таблиц HTML

The IBM Toolbox for Java HTMLTable class allows you to easily set up tables that you can use in HTML pages.

С его помощью можно работать со следующими атрибутами таблицы:

- Get and set the width of the border
- Get the number of rows in the table
- Add a column or row to the end of the table
- Remove a column or row at a specified column or row position

Пример: Применение классов HTMLTable

Ниже приведен пример применения классов HTMLTable:

“Пример: Применение классов HTMLTable” на стр. 624

Информация, связанная с данной

HTMLTable Javadoc

Класс HTMLTableCell:

The HTMLTableCell class takes any HTMLTagElement object as input and creates the table cell tag with the specified element. The element can be set on the constructor or through either of two setElement() methods.

Многие атрибуты ячейки можно восстановить или обновить с помощью методов класса HTMLTableCell. С помощью этих методов можно выполнять, например, следующие действия:

- Get or set the row span
- Get or set the cell height
- Set whether the cell data will use normal HTML line breaking conventions

Ниже приведен пример создания объекта HTMLTableCell и вывода тега:

```
//Создание объекта HTMLHyperlink.
HTMLHyperlink link = new HTMLHyperlink("http://www.ibm.com",
                     "Домашняя страница IBM");
HTMLTableCell cell = new HTMLTableCell(link);
cell.setHorizontalAlignment(HTMLConstants.CENTER);
System.out.println(cell.getTag());
```

The getTag() method above gives the output of the example:

<td align=″center″>IBM Home Page</td>

HTMLTableCell Javadoc

HTMLTagElement Javadoc

Класс HTMLTableRow:

The HTMLTableRow class creates a row within a table. Этот класс включает различные методы считывания и задания атрибутов строки.

The methods of the HTMLTableRow class allow you to:

- Add or remove a column from the row
- Get column data at the specified column Index
- Get column index for the column with the specified cell.
- Get the number of columns in a row
- Set horizontal and vertical alignments

Ниже приведен пример HTMLTableRow:

```
// Создание строки и установка выравнивания.
HTMLTableRow row = new HTMLTableRow();
row.setHorizontalAlignment(HTMLTableRow.CENTER);

// Создание и добавление информации о столбце к строке.
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));

// Добавление строки к объекту HTMLTable (предполагается, что этот объект уже существует).
table.addRow(row);
```

Информация, связанная с данной

HTMLTableRow Javadoc

Класс HTMLTableHeader:

The HTMLTableHeader class inherits from the HTMLTableCell class. It creates a specific type of cell, the header cell, giving you a **<th>** cell instead of a **<td>** cell. Как и для класса HTMLTableCell, вы можете применять различные методы для обновления или восстановления атрибутов ячейки заголовка.

Ниже приведен пример HTMLTableHeader:

```
// Создание заголовков таблицы.
HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("Счет"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("Имя"));
HTMLTableHeader balance_header = new HTMLTableHeader();
HTMLText balance = new HTMLText("БАЛАНС");
balance_header.setElement(balance);

// Добавление заголовков таблицы к объекту HTMLTable (предполагается, что этот объект уже существует).
table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);
```

Информация, связанная с данной

HTMLTableHeader Javadoc

Класс HTMLTableCaption:

The HTMLTableCaption class creates a caption for your HTML table. Этот класс содержит методы для обновления и восстановления атрибутов названия. For example, you can use the setAlignment() method to specify to which part of the table the caption should be aligned.

Ниже приведен пример HTMLTableCaption:

```
// Создание объекта HTMLTableCaption по умолчанию и текста заголовка.
HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Баланс счетов заказчиков - 1 января 2000 года");

// Добавление названия к объекту HTMLTable (предполагается, что этот объект уже существует).
table.setCaption(caption);
```

Информация, связанная с данной

HTMLTableCaption Javadoc

Класс текста HTML

The IBM Toolbox for Java HTMLText class allows you to access text properties for your HTML page. Using the HTMLText class, you can get, set, and check the status of many text attributes.

These attributes include the following:

- Get or set the size of the font
- Set the bold attribute on (true) or off (false) or determine if it is already on
- Set the underscore attribute on (true) or off (false) or determine if it is already on
- Get or set the horizontal alignment of the text

В следующем примере показано создание объекта HTMLText и выбор полужирного шрифта размера 5.

```
HTMLText text = new HTMLText("IBM");
text.setBold(true);
text.setSize(5);
System.out.println(text.getTag());
```

Последний оператор печати создаст следующий тег:

```
<font size="5"><b>IBM</b></font>
```

На странице HTML этот тег будет выглядеть следующим образом:

IBM

Информация, связанная с данной

HTMLText Javadoc

Классы HTMLTree

The HTMLTree class allows you to easily set up a hierarchical tree of HTML elements that you can use in HTML pages.

Помимо методов для работы с атрибутами дерева, этот класс содержит методы для:

- Get and set the HTTP Servlet Request
- Add an HTMLTreeElement or FileTreeElement to the tree
- Remove an HTMLTreeElement or FileTreeElement from the tree

Примеры: Применение классов HTMLTree

Различные способы применения классов HTMLTree продемонстрированы в следующих примерах:

- “Пример: Применение классов HTMLTree” на стр. 614
- “Пример: Создание просматриваемого дерева интегрированной файловой системы”

Информация, связанная с данной

HTMLTree Javadoc

Пример: Создание просматриваемого дерева интегрированной файловой системы:

Следующий пример состоит из трех файлов. Он демонстрирует процесс создания просматриваемого дерева интегрированной файловой системы. Для отображения объектов HTMLTree и FileListElement в сервлете используются фреймы.

- FileTreeExample.java - служит для создания фреймов HTML и запуска сервлета
- TreeNav.java - служит для создания и управления деревом
- TreeList.java - показывает содержимое объектов, выбранных в классе TreeNav.java

Класс HTMLTreeElement:

The HTMLTreeElement class represents an hierarchical element within an HTMLTree or other HTMLTreeElements.

Многие атрибуты этого дерева можно получить или изменить с помощью методов класса HTMLTreeElement. С помощью этих методов можно выполнять, например, следующие действия:

- Get or set the visible text of the tree element
- Get or set the URL for the expanded and collapsed icon
- Set whether the tree element will be expanded

Ниже приведен пример создания объекта HTMLTreeElement и вывода тега:

```
// Создание объекта HTMLTree.
HTMLTree tree = new HTMLTree();

// Создание родительского объекта HTMLTreeElement.
HTMLTreeElement parentElement = new HTMLTreeElement();
parentElement.setTextUrl(new HTMLHyperlink("http://myWebPage", "Моя Web-страница"));


// Создание дочернего объекта HTMLTreeElement.
HTMLTreeElement childElement = new HTMLTreeElement();
childElement.setTextUrl(new HTMLHyperlink("http://anotherWebPage", "Другая Web-страница"));
parentElement.addElement(childElement);

// Добавление элемента в иерархию.
tree.addElement(parentElement);
System.out.println(tree.getTag());
```

The getTag() method in the above example generates HTML tags like the following:

```
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>My Web Page</u></font></td>
</tr>

<tr>
<td> </td>
<td>
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>Another Web Page</u></font> </td>
</tr>
</table>
</td>
</tr>
</table>
```

Информация, связанная с данной

HTMLTreeElement Javadoc

Класс FileTreeElement:

The IBM Toolbox for Java FileTreeElement class represents the Integrated File System within an HTMLTree view.

Многие атрибуты этого дерева можно получить или изменить с помощью методов HTMLTreeElement. You can also get and set the name and path of NetServer shared drives.

Ниже перечислены некоторые действия, которые можно выполнить с помощью этих методов:

- Get or set the URL for the expanded and collapsed icon (inherited method)
- Set whether the tree element will be expanded (inherited method)
- Get or set the name of the NetServer shared drive
- Get or set the path of the NetServer shared drive

Пример: Применение класса FileTreeElement

Ниже приведен пример создания объекта FileTreeElement и вывода тега:

```
// Создание объекта HTMLTree.
HTMLTree tree = new HTMLTree();

// Создание объекта URLParser.
URLParser urlParser = new URLParser(httpServletRequest.getRequestURI());

// Создание объекта AS400.
AS400 system = new AS400(mySystem, myUserId, myPassword);

// Создание объекта IFSJavaFile.
IFSJavaFile root = new IFSJavaFile(system, "/QIBM");

// Создание объекта DirFilter и получение списка каталогов.
DirFilter filter = new DirFilter();
File[] dirList = root.listFiles(filter);

for (int i=0; i < dirList.length; i++)
{

    // Создание FileTreeElement.
    FileTreeElement node = new FileTreeElement(dirList[i]);

    // Настройка URL значка.
    ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
    sl.setHttpServletResponse(resp);
    element.setIconUrl(sl);

    // Добавление FileTreeElement к дереву.
    tree.addElement(element);
}

System.out.println(tree.getTag());
```

The getTag() method above gives the output of the example.

Информация, связанная с данной

FileTreeElement Javadoc

Класс FileListElement:

The IBM Toolbox for Java FileListElement class allows you to create a file list element, which represents the contents of an integrated file system directory.

С помощью объекта FileListElement можно представить содержимое общего диска a NetServer, получив и задав имя и путь к общим дискам a NetServer.

Класс FileListElement содержит методы, позволяющие:

- List and sort the elements of the file list
- Get and set the HTTP Servlet Request
- Get and set the FileListRenderer
- Get and set the HTMLTable with which to display the file list
- Get and set the name of a NetServer shared drive
- Get and set the path of a NetServer shared drive

Класс FileListElement можно применять совместно с другими классами из пакета HTML:

- С помощью класса FileListRenderer можно указать способ отображения списка файлов

- With the FileTreeElement class, you can create a traversable list of integrated file system files or NetServer shared files

Пример: Создание просматриваемого дерева интегрированной файловой системы с помощью класса FileListElement

The following example shows how you can use the FileListElement class with HTMLTree classes (FileTreeElement and HTMLTreeElement) to create a traversable integrated file system tree. Пример также содержит программу, позволяющую задавать путь к общему дереву NetServer.

"Пример: Создание просматриваемого дерева интегрированной файловой системы" на стр. 221

Информация, связанная с данной

FileListElement Javadoc

FileTreeElement Javadoc

HTMLTreeElement Javadoc

Класс FileListRenderer:

The IBM Toolbox for Java FileListRenderer class renders any field for File objects (directories and files) in a FileListElement.

Методы класса FileListRenderer позволяют выполнять следующие действия:

- Get the name of the directory
- Get the name of the file
- Get the name of the parent directory
- Return the row data that you want to display in the FileListElement

В этом примере продемонстрировано создание объекта FileListElement с помощью объекта преобразования:

```
// Создание объекта FileListElement.
FileListElement fileList = new FileListElement(sys, httpservletrequest);

// Настройка объекта преобразования для этого сервлета, расширяющего
// класс FileListRenderer и переопределяющего соответствующие методы.
fileList.setRenderer(new myFileListRenderer(request));
```

Если вы не хотите применять объект преобразования по умолчанию, класс FileListRenderer можно расширить, переопределив его методы и добавив новые. Предположим, например, что имена каталогов или файлов с определенными расширениями не должны передаваться в объект FileListElement. После расширения класса и переопределения соответствующего метода вместо этих файлов и каталогов будет возвращаться значение null, в результате чего эти файлы и каталоги не будут показаны.

To fully customize the rows within a FileListElement, use the getRowData() method. Например, с его помощью можно добавить столбец и изменить порядок столбцов. Если вам достаточно функций объекта FileListRenderer по умолчанию, то ничего изменять не нужно, так как класс FileListElement создает объект преобразования FileListRenderer по умолчанию.

Ссылки, связанные с данной

"Класс FileListElement" на стр. 223
The IBM Toolbox for Java FileListElement class allows you to create a file list element, which represents the contents of an integrated file system directory.

Информация, связанная с данной

FileListRenderer Javadoc

Классы ReportWriter

The com.ibm.as400.util.reportwriter package provides classes that enable you to easily access and format data from an XML source file or data produced by servlets or JavaServer Pages.

Под пакетом reportwriter понимаются три разных, но связанных между собой пакета:

- com.ibm.as400.util.reportwriter.pclwriter
- com.ibm.as400.util.reportwriter.pdfwriter
- com.ibm.as400.util.reportwriter.processor

Эти пакеты содержат различные классы, позволяющие форматировать потоки данных XML и создавать отчеты в этих форматах. Убедитесь, что в переменной CLASSPATH заданы необходимые файлы jar, включая анализатор XML и обработчик XSLT. Дополнительная информация приведена на следующих страницах:

- Файлы Jar

Классы контекста из пакетов pclwriter и pdfwriter содержат методы, которые необходимы классам ReportProcessor для преобразования данных XML и JSP в выбранный формат:

- Класс PCLContext совместно с классом ReportWriter применяется для создания отчета в формате Управляющий язык принтера (PCL) Hewlett Packard.
- Use PDFContext in combination with a ReportWriter class to generate a report in the Adobe Portable Document Format (PDF).

ReportProcessor classes (in the processor package) enable you to generate formatted reports from information your application gathers from Java servlets and JavaServer Pages (JSPs).

- Класс JSPReportProcessor предназначен для получения данных от сервлетов и страниц JSP и создания отчетов в доступных форматах (контекстах).

Классы Context

The IBM Toolbox for Java context classes support specific data formats, that, in combination with the OutputQueue and SpooledFileOutputStream classes, enable the ReportWriter classes to generate reports in that format and put those reports in a spool file.

Приложению нужно только лишь создать экземпляр класса Context. После этого классы ReportWriter будут использовать этот экземпляр для создания отчетов. Приложение не должно напрямую вызывать никакие методы из какого-либо класса Context. Методы PCLContext и PDFContext предназначены для внутреннего применения классами ReportWriter.

Для создания экземпляра класса Context требуются классы OutputStream (из пакета java.io) и PageFormat (из пакета java.awt.print). Example: Using JSPReportProcessor with PDFContext show how you can construct and use the Context classes with other ReportWriter classes to generate reports.

OutputQueue Javadoc

SpooledFileOutputStream Javadoc

“Классы ReportWriter”
The com.ibm.as400.util.reportwriter package provides classes that enable you to easily access and format data from an XML source file or data produced by servlets or JavaServer Pages.

“Пример: Применение XSLReportProcessor с PCLContext” на стр. 639
This example should not be used as the XSLReportProcessor class is no longer supported.

“Пример: Работа с классом JSPReportProcessor с помощью PDFContext” на стр. 635
This example uses the JSPReportProcessor and the PDFContext classes to obtain data from a specified URL and convert the data to the PDF format. The data is then streamed to a file as a PDF document.

Класс JSPReportProcessor

The JSPReportProcessor class enables you to create a document or report from the contents of a JavaServer Page (JSP) or Java servlet.

Этот класс позволяет получить JSP или сервлет с указанным адресом и создать документ на основе его содержимого. JSP или сервлет предоставляет данные для документа, включая объекты форматирования XSL. Перед созданием страниц документа необходимо задать контекст вывода и источник входных данных JSP. Данные отчета можно преобразовать в указанный формат вывода.

Класс JSPReportProcessor позволяет выполнять следующие операции:

- Process the report
- Set a URL as the template

Ниже приведены примеры применения классов JSPReportProcessor и PDFContext для создания отчета. Примеры программ приведены как для Java, так и для JSP; их можно просмотреть с помощью следующих ссылок. You can also download a ZIP file that contains the example JSP, XML, and XSL source files for the JSPReportProcessor examples:

- “Пример: Работа с классом JSPReportProcessor с помощью PDFContext” на стр. 635
- “Пример: Файл JSP для класса JSPReportProcessor” на стр. 637

 Java Server Pages technology

Класс XSLReportProcessor

The XSLReportProcessor class is no longer supported and should not be used.

The XSLReportProcessor class enables you to create a document or report by transforming and formatting your XML source data using an XSL stylesheet. С помощью этого класса можно создать отчет на базе формы XSL, содержащей необходимые объекты форматирования XSL. Затем данные отчета можно преобразовать в нужный формат потока данных с помощью класса Context.

С помощью класса XSLReportProcessor можно выполнять следующие операции:

- Set the XSL stylesheet
- Set the XML data source
- Set the XSL FO source
- Process a report

Примеры

Ниже приведены примеры создания отчетов с помощью классов XSLReportProcessor и PCLContext. Примеры содержат код Java, XML и XSL; примеры кода можно просмотреть с помощью приведенных ниже ссылок. Можно также загрузить архивный файл ZIP, который содержит примеры применения классов JSPReportProcessor и XSLReportProcessor для JSP, XML и XSL:

- Пример: Применение класса XSLReportProcessor с PCLContext
- Пример: Файл XML для XSLReportProcessor
- Пример: Файл XSL для XSLReportProcessor

For more information about XML and XSL, see the XML Toolkit topic in the Information Center.

Классы ресурсов

Пакет Resource и его классы устарели. Рекомендуется использовать пакет Access.

Пакет com.ibm.as400.resource обеспечивает общую среду для работы с различными объектами и списками AS400.Эта среда является согласованным программным интерфейсом для всех таких объектов и списков.

В пакет ресурсов входят следующие классы:

R • Resource - an object that represents a system resource, such as a user, printer, job, message, or file.
R Действительные подклассы resource:

R – RIFSFile
R – RJavaProgram
R – RJob
R – RPrinter
R – RQueuedMessage
R – RSoftwareResource
R – RUser

R **Примечание:** Классы NetServer в пакете доступа также являются действительными подклассами класса
R Resource.

R • ResourceList - an object that represents a list of system resources, such as a list of users, printers, jobs, messages, or
R files. Действительные подклассы resource:
R – RIFSFileList
R – RJobList
R – RJobLog
R – RMessageQueue
R – RPrinterList
R – RUserList

- Presentation - объект, отвечающий за представление информации об объектах ресурсов, списках ресурсов, атрибутах, выбранных значениях и отсортированных списках конечным пользователям

Классы Resource и ChangeableResource

Пакет Resource и его классы устарели. Рекомендуется использовать пакет Access.

R The com.ibm.as400.resource.Resource and com.ibm.as400.resource.ChangeableResource abstract classes represent
R aSystem i resource.

Класс Resource

Абстрактный класс Resource обеспечивает общий доступ к атрибутам любого ресурса. Каждому атрибуту соответствует ИД, и каждый подкласс класс Resource содержит информацию об ИД атрибутов, которые он поддерживает.

Класс Resource позволяет только получать значения атрибутов, но не изменять их.

R Продукт IBM Toolbox for Java содержит следующие объекты ресурсов:

- RIFSFile - represents a file or directory in the integrated file system
- RJavaProgram - represents a Java program on the server
- RJob - represents a System i job
- RPrinter - represents a System i printer
- RQueuedMessage - represents a message in a System i message queue or job log
- RSoftwareResource - represents a licensed program on the system
- RUser - represents a System i user

Класс ChangeableResource

R The ChangeableResource abstract class, a subclass of Resource, adds the ability to change attribute values of a system
R resource. Измененные значения атрибутов заносятся во внутренний кэш и хранятся в нем, пока не будут
R зафиксированы или отменены. Это позволяет изменять значения нескольких атрибутов одновременно.

Примечание: Классы NetServer в пакете доступа также являются действительными подклассами класса Resource и ChangeableResource.

Примеры

Ниже приведены примеры применения действительных классов, дочерних по отношению к классам Resource и ChangeableResource, а также примеры работы с подклассами абстрактных классов Resource и ChangeableResource.

- Получение значения атрибута из класса RUser, дочернего по отношению к классу Resource
- Изменение значений атрибута с помощью класса RJob, дочернего по отношению к классу ChangeableResource
- Доступ к ресурсам с помощью общего кода

Списки ресурсов

Пакет Resource и его классы устарели. Рекомендуется использовать пакет Access.

R The com.ibm.as400.resource.ResourceList class represents a list of system resources. Этот абстрактный класс
R обеспечивает общий доступ к информации списка.

R Продукт IBM Toolbox for Java содержит следующие списки ресурсов:

- RIFSFileList - represents a list of files and directories in the integrated file system
- RJobList - represents a list of system jobs
- RJobLog - represents a list of messages in a system job log
- RMessageQueue - represents a list of messages in a system message queue
- RPrinterList - represents a list of system printers
- RUserList - represents a list of system users

Список ресурсов может находится в двух состояниях: открытом и закрытом. Для работы с содержимым списка его необходимо открыть. Для обеспечения быстрого доступа к информации списка и эффективного управления оперативной памятью большинство списков выводится на экран во время загрузки.

Списки ресурсов позволяют:

- Open the list
- Close the list
- Access a specific Resource from the list
- Wait for a particular resource to load
- Wait for the complete resource list to load

Значения выбора позволяют фильтровать списки ресурсов. Каждому значению выбора соответствует ИД выбора. Похожая методика применяется для сортировки списков ресурсов с помощью значений сортировки. Каждому значению сортировки соответствует ИД сортировки. Подклассы, дочерние по отношению к классу ResourceList, обычно содержат информацию о поддерживаемых ИД сортировки и выбора.

Примеры

Ниже приведены примеры различных приемов работы со списками ресурсов:

- Пример: Получение и печать содержимого объекта ResourceList
- Пример Обращение к объекту ResourceList с помощью общего кода
- Пример: Представление списка ресурсов в сервлете (таблица HTML)

Отказ от гарантий на предоставляемый код

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.
>
> Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Класс Presentation

Пакет Resource и его классы устарели. Рекомендуется использовать пакет Access.

Каждому объекту ресурса, списку ресурсов и объекту мета-данных соответствует объект com.ibm.as400.resource.Presentation, обеспечивающий преобразование информации, например, имени, полного имени или значка.

Пример: Печать списка ресурсов и его значений сортировки с помощью объектов Presentation

Информация объекта Presentation позволяет представлять в текстовом формате, удобном для конечных пользователей, объекты ресурсов, списков ресурсов, атрибутов, отфильтрованных и отсортированных списков.

```
void printCurrentSort(ResourceList resourceList) throws ResourceException
{
    // Получить представление для объекта ResourceList и напечатать его полное имя.
    Presentation resourceListPresentation = resourceList.getPresentation();
    System.out.println(resourceListPresentation.getFullName());

    // Получить текущее значение сортировки.
    Object[] sortIDs = resourceList.getSortValue();

    // Напечатать все ИД сортировки.
    for(int i = 0; i < sortIDs.length; ++i)
    {
        ResourceMetaData sortMetaData = resourceList.getSortMetaData(sortIDs[i]);
        System.out.println("Сортировка по " + sortMetaData.getName());
    }
}
```

Классы защиты

Классы защиты IBM Toolbox for Java позволяют установить защищенное соединение с сервером, идентифицировать пользователя и связать его с нитью операционной системы локального сервера.

Предусмотрены следующие службы защиты:

- Инфраструктура связи Java Secure Socket Extension (JSSE), обеспечивающая защиту соединений как с помощью шифрования данных сеанса клиент-сервер, так и с помощью идентификации на сервере.
- Службы идентификации предназначены для выполнения следующих задач:
  - Идентификации пользователя путем сравнения его имени и пароля со значениями, заданными при регистрации в i5/OS.
  - Замены владельца текущей нити i5/OS.

SSL

Secure Sockets Layer (SSL) обеспечивает защиту соединений как с помощью шифрования данных сеанса клиент-сервер, так и с помощью идентификации на сервере.

Применение SSL приводит к снижению производительности, поскольку скорость передачи данных по защищенному соединению ниже, чем по соединению без шифрования данных. Соединения SSL должны применяться только в том случае, когда защита данных важнее скорости передачи: например, при передаче номеров кредитных карт или операциях с банковскими счетами.

За дополнительной информацией обратитесь в местное представительство фирмы IBM.

Using encryption between the IBM Toolbox for Java classes and the i5/OS servers

Применение SSL для шифрования данных, передаваемых между IBM Toolbox for Java и серверами i5/OS:

Для шифрования данных, передаваемых между классами IBM Toolbox for Java и серверами i5/OS, может применяться SSL.

На стороне клиента включите JSSE для шифрования данных. Для настройки функции шифрования данных на серверах i5/OS применяется Диспетчер цифровых сертификатов i5/OS.

Настройка применения SSL на сервере и на клиенте

Для шифрования данных, передаваемых между классами IBM Toolbox for Java и серверами i5/OS, выполните следующие действия:

1. Настройте серверы для работы с зашифрованными данными.
2. Включите шифрование данных в IBM Toolbox for Java с помощью объекта SecureAS400.

   Примечание: После выполнения первых двух шагов будет создано защищенное соединение между сервером и клиентом. Для передачи данных по нему приложение должно работать с объектом SecureAS400 IBM Toolbox for Java. Будет выполняться шифрование только тех данных, которые передаются через объект SecureAS400. При работе с объектом AS400 данные не шифруются, и для их передачи применяется обычное соединение.

Setting up System i5 to use SSL:

To set up your system to use SSL with IBM Toolbox for Java, complete the following steps.

1. Install the following to your System i:

   Примечание: This step is necessary only for systems running i5/OS versions prior to V5R4. В V5R4 и последующих выпусках продукт IBM Cryptographic Access Provider не поставляется.

   - IBM Cryptographic Access Provider (128-разрядная версия) для iSeries, 5722-AC3. Этот продукт обеспечивает поддержку шифрования для серверных приложений.
2. Получите и настройте сертификат сервера.
3. Apply the certificate to the following systems that are used by IBM Toolbox for Java:
   - QIBM_OS400_QZBS_SVR_CENTRAL
   - QIBM_OS400_QZBS_SVR_DATABASE
   - QIBM_OS400_QZBS_SVR_DTAQ
   - QIBM_OS400_QZBS_SVR_NETPRT
   - QIBM_OS400_QZBS_SVR_RMTCMD
   - QIBM_OS400_QZBS_SVR_SIGNON
   - QIBM_OS400_QZBS_SVR_FILE

- QIBM_OS400_QRW_SVR_DDM_DRDA

Получение и настройка сертификатов сервера

Перед получением и установкой сертификата сервера необходимо установить следующие продукты:

- IBM HTTP Server (5761-DG1) licensed program
- Компонент 34 Базовой операционной системы (Диспетчер цифровых сертификатов)

Процесс получения и установки сертификата сервера зависит от типа сертификата:

- Если сертификат получен от уполномоченной сертификатной компании (такой как VeriSign, Inc. или RSA Data Security, Inc.), install the certificate on the system then apply it to the host servers.
- If you choose not to use a certificate from a trusted authority, you can build your own certificate to be used on the system. Создайте сертификат с помощью Диспетчера цифровых сертификатов:
  1. Create the certificate authority on the system. See the Information Center topic Acting as your own CA.
  2. Создайте в созданной сертификатной компании сертификат системы.
  3. Укажите серверы, которые будут применять созданный сертификат системы.

Службы идентификации

В IBM Toolbox for Java предусмотрены классы, взаимодействующие со службами защиты i5/OS.

В частности, поддерживается идентификация пользователей, или *субъектов*, путем сравнения их имен и паролей со значениями, заданными при регистрации в i5/OS. После идентификации пользователю может быть выдано одноразовое разрешение. С помощью этого разрешения можно изменить владельца текущей нити i5/OS, переключившись на идентифицированного пользователя. С точки зрения работы нити, такая смена владельца эквивалентна входу пользователя в систему.

Предоставляемая поддержка

The AS400 object provides authentication for a given user profile and password against the server. Идентифицированному пользователю могут быть выданы разрешения и паспорта Kerberos.

Примечание: Для использования паспортов Kerberos необходимо установить J2SDK версии 1.4 и настроить интерфейс прикладных программ Общая служба защиты Java (JGSS). For more information about JGSS, see the J2SDK, v1.4 Security Documentation .

Для применения паспортов Kerberos в объекте AS400 необходимо задать только имя системы (не указывая пароль). Идентификация пользователя выполняется средствами JGSS. В каждый момент времени объект AS400 может использовать только один способ идентификации. Если вы зададите пароль, то разрешение или паспорт Kerberos будут удалены.

To use profile tokens, use the getProfileToken() methods to retrieve instances of the ProfileTokenCredential class. Такое разрешение представляет идентифицированный профайл пользователя и его пароль на конкретном сервере. Разрешение действительно в течение определенного времени, обычно не более одного часа. Однако в некоторых случаях срок действия разрешения можно продлить.

Примечание: При применении класса ProfileTokenCredential ознакомьтесь с различными способами создания разрешений, приведенными в конце этого документа.

В примере ниже создается объект системы, с помощью которого генерируется кратковременное разрешение. Затем на основе разрешения создается еще один объект системы, и этот объект применяется для подключения к службе выполнения команд:

```
AS400 system = new AS400("mySystemName", "MYUSERID", "MYPASSWORD");
ProfileTokenCredential myPT = system.getProfileToken();
AS400 system2 = new AS400("mySystemName", myPT);
system2.connectService(AS400.COMMAND);
```

Изменение владельца нити

Разрешение может получить как локальный, так и удаленный пользователь. После создания такого разрешения оно может быть преобразовано приложением в поток байт или передано другому процессу. В частности, такое разрешение может быть передано процессу системы server для *замены* владельца нити i5/OS. После этого все действия будут выполняться от имени идентифицированного пользователя.

Такая поддержка в первую очередь предназначена для двухуровневых приложений, в которых идентификация пользователя с помощью пароля выполняется графическим пользовательским интерфейсом на первом уровне (на PC), а все действия для этого пользователя выполняются на втором уровне (на сервере). Применение класса ProfileTokenCredentials позволяет избежать передачи ИД и пароля пользователя по сети. В данном случае программе на втором уровне передается разрешение, выданное профайлу пользователя. После этого программа может вызвать функцию *swap()* и работать в системе i5/OS от имени этого пользователя.

Примечание: Несмотря на то, что в силу ограниченности интервала действия последний способ является более безопасным, чем передача пользовательского профайла и пароля, разрешения должны обрабатываться приложениями как конфиденциальные данные. Поскольку разрешение служит эквивалентом имени и пароля пользователя, оно может быть использовано другим приложением с целью получить доступ от имени этого пользователя. Вся ответственность за защиту такого разрешения ложится на само приложение.

Методы создания разрешений класса ProfileTokenCredential

При работе с методами создания разрешений класса ProfileTokenCredential необходимо учитывать, что пароли могут задаваться по-разному:

- В виде специального значения, такого как *NOPWD или *NOPWDCHK, - с помощью заданного специального целого значения
- В виде пароля пользовательского профайла - с помощью объекта String, задающего пароль

Примечание: В IBM Toolbox for Java выпуска V5R3 не включены методы setToken, для которых не нужно задавать способ указания пароля.

Кроме того, методы setToken позволяют удаленным пользователям задавать специальные значения паролей и поддерживают пароли длиной до 128 символов.

Задать специальное целое значение пароля, такое как *NOPWD или *NOPWDCHK, можно с помощью одного из следующих методов:

- setToken(AS400Principal субъект, int Специальное_значение_пароля)
- setToken(String имя, int Специальное_значение_пароля)

Класс ProfileTokenCredential содержит следующие статические константы для специальных целых значений паролей:

- ProfileTokenCredential.PW_NOPWD: соответствует *NOPWD
- ProfileTokenCredential.PW_NOPWDCHK: соответствует *NOPWDCHK

Задать пароль пользовательского пароля с помощью объекта String позволяют следующие методы:

- setTokenExtended(AS400Principal субъект, String пароль)
- setTokenExtended(String имя, String пароль)

Методы setTokenExended не позволяют передавать специальные строчные значения паролей в качестве параметров. Например, они не поддерживают строчные значения паролей *NOPWD.

For more information, see the ProfileTokenCredential Javadoc reference information.

Пример

Ознакомьтесь с примером кода, в котором профайлу пользователя выдается одноразовое разрешение, позволяющее заменить владельца нити i5/OS и работать в системе от имени указанного пользователя.

AS400 Javadoc

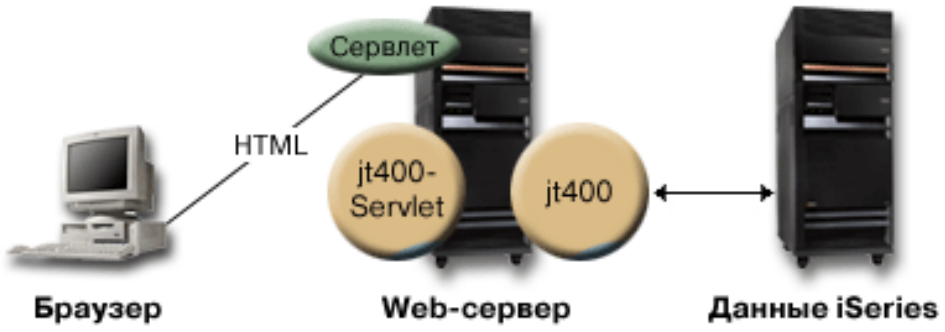
ProfileTokenCredential Javadoc

Классы сервлетов

The servlet classes that are provided with IBM Toolbox for Java work with the access classes, which are located on the Webserver, to give you access to information located on the server. Вы можете применять классы сервлетов по своему усмотрению в разрабатываемых проектах.

The following diagram shows how the servlet classes work between the browser, webserver, and System i5 data. Браузер подключается к Web-серверу, на котором запущен сервлет. Файлы jt400Servlet.jar и jt400.jar размещены на Web-сервере, поскольку классы сервлетов получают данные с помощью классов доступа и представляют данные с помощью классов HTML. The webserver is connected to the server where the data is.

Рисунок 1: Принцип работы сервлетов



“Подробное описание рисунка 1: Работа сервлетов (rzahh585.gif)”

Примечание: Файл jt400Servlet.jar содержит классы HTML наряду с классами сервлетов. Для работы с классами из пакетов com.ibm.as400.util.html и com.ibm.as400.util.servlet необходимо добавить файлы jt400Servlet.jar и jt400.jar в переменную CLASSPATH.

Дополнительные источники информации о сервлетах перечислены в разделе ссылок на справочную информацию.

Подробное описание рисунка 1: Работа сервлетов (rzahh585.gif)

found in IBM Toolbox for Java: Servlet classes

Данный рисунок иллюстрирует общий принцип работы сервлетов.

Описание

Рисунок состоит из следующих компонентов:

- Изображения слева персонального компьютера, отмеченного надписью ’Браузер’ и представляющего экземпляр браузера на PC.
- An image of an System i on the right, labeled ’System i Data,’ that represents the location of the data that you want the servlet to access.
- An image of aSystem i in the middle (between the other two images), labeled ’WebServer,’ that represents the Web server. Несколько фигур внутри образа Web-сервера обозначают файлы и функции Web-сервера:
  - Зеленый овал с именем Сервлет представляет расположение кода сервлета.
  - Желтый круг с именем jt400Servlet представляет расположение файла jt400Servlet.jar.
  - Желтый круг с именем jt400 представляет расположение файла jt400.jar.

Примечание: The WebServer does not have to be on a System i, but it can be, and can even be the same server as that indicated by the System i Data image.

- Линий, соединяющих изображения.

Линия с надписью HTML соединяет браузер (изображение слева) с сервлетом (зеленый овал) на Web-сервере (изображение посередине). Линия отмечена надписью HTML, поскольку чаще всего сервлеты передают браузерам данные в формате HTML.

На Web-сервере работают два файла Jar IBM Toolbox for Java (желтые круги): jt400Servlet.jar и jt400.jar. The classes in jt400Servlet.jar, along with the classes in jt400.jar, enable the WebServer to run a servlet that easily connects to servers that contain System iData (the right image). Соответствующее соединение обозначено двунаправленной стрелкой.

Классы идентификации

Two classes in the servlet package perform authentication for servlets, AuthenticationServlet and AS400Servlet.

Класс AuthenticationServlet

AuthenticationServlet is an HttpServlet implementation that performs basic authentication for servlets. В подклассах класса AuthenticationServlet должен быть переопределен один или несколько следующих методов:

- Override the validateAuthority() method to perform the authentication (required)
- Override the bypassAuthentication() method so that the subclass authenticates only certain requests
- Override the postValidation() method to allow additional processing of the request after authentication

Методы класса AuthenticationServlet позволяют:

- Initialize the servlet
- Get the authenticated user ID
- Set a user ID after bypassing authentication
- Log exceptions and messages

Класс AS400Servlet

The AS400Servlet class is an abstract subclass of AuthenticationServlet that represents an HTML servlet. Для совместного использования соединений и управления числом соединений пользователя сервлета с сервером применяется пул соединений.

Методы класса AS400Servlet позволяют:

- Validate user authority (by overriding the validateAuthority() method of the AuthenticationServlet class)
- Connect to a system
- Get and return connection pool objects to and from the pool
- Close a connection pool
- Get and set the HTML document head tags

- Get and set the HTML document end tags

Дополнительные источники информации о сервлетах перечислены в разделе ссылок на справочную информацию.

AuthenticationServlet Javadoc

AS400Servlet Javadoc

Класс RowData

The RowData class is an abstract class that provides a way to describe and access a list of data.

Классы RowData позволяют:

- Получать и устанавливать текущую позицию
- Get the row data at a given column using the getObject() method
- Get the meta data for the row
- Get or set the properties for an object at a given column
- Get the number of rows in the list using the length() method.

Позиция RowData

Существуют несколько методов для определения и настройки текущей позиции в списке. Ниже перечислены соответствующие методы для классов RowData.

Указание позиции		**Определение позиции**
absolute()	next()	getCurrentPosition()
afterLast()	previous()	isAfterLast()
beforeFirst()	relative()	isBeforeFirst()
first()		isFirst()
last()		isLast()

Информация, связанная с данной

RowData Javadoc

Класс ListRowData:

The IBM Toolbox for Java ListRowData class represents a list of data in table form. In the table, each row contains a finite number of columns determined by the ListMetaData object and each column within a row contains an individual data item. The data can be a directory in the integrated file system, a list of jobs, a list of printers, or a variety of other data.

Класс ListRowData позволяет выполнить следующие операции:

- Add and remove rows to and from the result list.
- Get and set the row
- Get information about the list's columns with the getMetaData() method
- Set column information with the setMetaData() method

The ListRowData class represents a list of data. ListRowData can represent many types of information, including the following, through IBM Toolbox for Java access classes:

- Каталог интегрированной файловой системы
- Список заданий
- Список сообщений из очереди сообщений
- Список пользователей

- Список принтеров
- Список буферных файлов

Пример

Ниже приведен пример работы классов ListRowData и HTMLTableConverter. Пример содержит программу на Java, код HTML и вид Web-страницы.

“Пример: Работа с ListRowData” на стр. 658

ListRowData Javadoc

Класс RecordListRowData:

The IBM Toolbox for Java RecordListRowData class allows you to do the following:
- Add and remove rows to and from the record list.
- Get and set the row
- Set the record format with the setRecordFormat method
- Get the record format

The RecordListRowData class represents a list of records. Запись можно получить с сервера в одном из следующих форматов:
- A record to be written to or read from a server file
- An entry in a data queue
- The parameter data from a program call
- В любом другом формате сервера, который необходимо преобразовать в формат Java

Порядок использования классов RecordListRowData и HTMLTableConverter иллюстрируется примером. Он включает программу на Java, соответствующий код HTML и пример Web-страницы.

Информация, связанная с данной

RecordListRowData Javadoc

Класс ResourceListRowData:

The IBM Toolbox for Java ResourceListRowData class represents a resource list of data. Use ResourceListRowData objects to represent any implementation of the ResourceList interface.

Списки ресурсов представляются в виде набора строк, в котором каждое строка содержит конечное число столбцов. Число столбцов определяется по числу ИД атрибутов столбцов. Каждый столбец в строке содержит отдельный элемент данных.

Класс ResourceListRowData содержит методы, позволяющие выполнять следующие операции:
- Get and set column attribute IDs
- Get and set the resource list
- Retrieve the number of rows in the list
- Get the column data for the current row
- Get the property list of the data object
- Get the metadata for the list

Пример: Представление списка ресурсов в сервлете

Отказ от гарантий на предоставляемый код

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.
>
> Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Ссылки, связанные с данной

“Списки ресурсов” на стр. 228
Пакет Resource и его классы устарели. Рекомендуется использовать пакет Access.

Информация, связанная с данной

ResourceListRowData Javadoc

Класс SQLResultSetRowData:

The SQLResultSetRowData class represents an SQL result set as a list of data. This data is generated by an SQL statement through JDBC. With methods provided, you can get set the result set metadata.

В приведенном примере проиллюстрировано применение классов ListRowData и HTMLTableConverter. Он содержит программу на Java, соответствующий код HTML и вид Web-страницы.

Ссылки, связанные с данной

“Классы JDBC” на стр. 61
JDBC - это интерфейс прикладных программ (API), который входит в пакет Java и позволяет программам на Java работать с широким спектром баз данных.

Информация, связанная с данной

SQLResultSetRowData Javadoc

Классы RowMetaData

The RowMetaData class defines an interface that you use to find out information about the columns of a RowData object.

Классы RowMetaData позволяют выполнять следующие операции:

- Get the number of columns
- Get the name, type, or size of the column
- Get or set the column label
- Get the precision or scale of the column data
- Determine if the column data is text data

Информация, связанная с данной

RowMetaData Javadoc

Класс ListMetaData:

The IBM Toolbox for Java ListMetaData class lets you get information about and change settings for the columns in a ListRowData class. It uses the setColumns() method to set the number of columns, clearing any previous column information. Кроме того, число столбцов можно указать в конструкторе.

Пример

Ниже приведен пример работы классов ListMetaData, ListRowData и HTMLTableConverter. Он содержит программу на Java, соответствующий код HTML и вид Web-страницы.

“Пример: Работа с ListRowData” на стр. 658

ListMetaData Javadoc

“Класс ListRowData” на стр. 235
The IBM Toolbox for Java ListRowData class represents a list of data in table form. In the table, each row contains a finite number of columns determined by the ListMetaData object and each column within a row contains an individual data item. The data can be a directory in the integrated file system, a list of jobs, a list of printers, or a variety of other data.

Класс RecordFormatMetaData:

The RecordFormatMetaData makes use of the IBM Toolbox for Java RecordFormat class. It allows you to provide the record format when you set the constructor’s parameters or use the get set methods to access the record format.

Ниже приведен пример создания объекта класса RecordFormatMetaData:

```
// Создание объекта RecordFormatMetaData из формата записи последовательного файла.
RecordFormat recordFormat = sequentialFile.getRecordFormat();
RecordFormatMetaData metadata = new RecordFormatMetaData(recordFormat);

// Вывод имен столбцов файла.
int numberOfColumns = metadata.getColumnCount();
for (int column=0; column < numberOfColumns; column++)
{
   System.out.println(metadata.getColumnName(column));
}
```

Информация, связанная с данной

RecordFormatMetaData Javadoc

RecordFormat Javadoc

Класс SQLResultSetMetaData:

The SQLResultSetMetaData class returns information about the columns of an SQLResultSetRowData object. You can either provide the result set when you set the constructor’s parameters or use the get and set methods to access the result set meta data.

Ниже приведен пример создания объекта класса SQLResultSetMetaData:

```
// Создание объекта SQLResultSetMetaData на основе метаданных набора результатов.
SQLResultSetRowData rowdata = new SQLResultSetRowData(resultSet);
SQLResultSetMetaData sqlMetadata  = rowdata.getMetaData();

// Просмотр значений точности числовых полей
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("Столбец: " + name + " содержит символьные данные.");
}
   else
   {
```

```
      System.out.println("Столбец: " + name + " содержит числа с точностью " +
                         sqlMetadata.getPrecision(column));
   }
}
```

SQLResultSetMetaData Javadoc

“Класс SQLResultSetRowData” на стр. 237
The SQLResultSetRowData class represents an SQL result set as a list of data. This data is generated by an SQL statement through JDBC. With methods provided, you can get set the result set metadata.

Классы преобразования

You use the IBM Toolbox for Java converter classes to convert row data into formatted string arrays.

Результатом будет представление в формате HTML, готовое к использованию на странице HTML. Преобразование выполняется следующими классами:

Класс StringConverter:

The StringConverter class is an abstract class that represents a row data string converter. It provides a convert() method to convert row data. Этот метод возвращает одномерный массив, в который была преобразована строка данных.

Информация, связанная с данной

StringConverter Javadoc

Класс HTMLFormConverter:

The IBM Toolbox for Java HTMLFormConverter classes extend StringConverter by providing an additional convert method called convertToForms(). Этот метод позволяет преобразовать данные из записей в массив однострочных таблиц HTML. Соответствующие теги позволяют выводить форматированную информацию в браузере.

Внешний вид формы HTML можно изменить с помощью набора методов, позволяющих просматривать и изменять атрибуты формы. В частности, можно изменить следующие атрибуты:

- Alignment
- Cell spacing
- Header hyper links
- Ширина

Пример: Применение класса HTMLFormConverter

Ниже приведен пример использования класса HTMLFormConverter. (Этот пример можно откомпилировать и запустить на работающем Web-сервере.)

Применение класса HTMLFormConverter

Ссылки, связанные с данной

“Класс StringConverter”
The StringConverter class is an abstract class that represents a row data string converter. It provides a convert() method to convert row data. Этот метод возвращает одномерный массив, в который была преобразована строка данных.

Информация, связанная с данной

HTMLFormConverter Javadoc

Класс HTMLTableConverter:

The HTMLTableConverter class extends StringConverter by providing a convertToTables() method. Этот метод служит для преобразования строковых данных в массив таблиц HTML, которые сервлет может применять для вывода списка в браузере.

You can use the getTable() and setTable() methods to choose a default table that will be used during conversion. You can set table headers within the HTML table object or you can use the meta data for the header information by setting setUseMetaData() to true.

The setMaximumTableSize() method allows you to limit the number of rows in a single table. Если строковые данные не помещаются в таблицу указанного размера, то преобразователь создаст в массиве вывода другую таблицу HTML. Эта процедура будет продолжаться до тех пор, пока не будут преобразованы все строковые данные.

Примеры

Ниже приведены примеры использования класса HTMLTableConverter:

- Пример: Работа с ListRowData
- Пример: Работа с RecordListRowData
- Пример: Работа с SQLResultSetRowData
- Пример: Применение класса ResourceList в сервлете

Отказ от гарантий на предоставляемый код

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.
>
> Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

HTMLTableConverter Javadoc

Классы Utility

The utility classes enable you to do administrative tasks, such as using the AS400JarMaker class.

В >IBM Toolbox for Java предусмотрены следующие утилиты:

Установка и обновление классов на клиенте

For most installation and update purposes, the IBM Toolbox for Java classes can be referenced at their location in the integrated file system on the server.

Так как к этому расположению применяются временные исправления программ (PTF), программы на Java , обращающиеся к классам непосредственно на сервере, получают обновления автоматически. But, accessing the classes from the server does not always work, specifically for the following situations:

- Если рабочая станция подключена к серверу по линии связи с низким быстродействием, то загрузка классов с сервера на рабочую станцию будет выполняться слишком медленно.
- If Java applications use the CLASSPATH environment variable to access the classes on the client file system, you need System i Access for Windows to redirect file system calls to the server. It may not be possible for System i Access for Windows to reside on the client.

В перечисленных выше случаях оптимальным вариантом является установка классов в клиентской системе.

AS400ToolboxJarMaker

Формат файлов JAR был специально разработан для ускорения загрузки файлов программ на Java. Класс AS400ToolboxJarMaker еще больше сокращает время загрузки путем сокращения размера файла JAR IBM Toolbox for Java.

Кроме этого, класс AS400ToolboxJarMaker позволяет распаковывать файлы JAR с целью получить доступ к их отдельным компонентам.

Гибкость класса AS400ToolboxJarMaker

Все функции работы с архивами JAR реализованы с помощью класса JarMaker и его подкласса AS400ToolboxJarMaker:

- Более общий инструмент JarMaker предназначен для работы с любыми файлами JAR и ZIP. Он позволяет разбивать архивы и сокращать их размер путем удаления ненужных классов.
- The AS400ToolboxJarMaker customizes and extends JarMaker functions for easier use with IBM Toolbox for Java JAR files.

По вашему усмотрению вы можете использовать методы AS400ToolboxJarMaker в программе на Java или вызывать их как независимые приложения из командной строки. Для вызова AS400ToolboxJarMaker из командной строки введите:

```
java utilities.JarMaker [опции]
```

где

- опции - одна или несколько опций

For a complete set of options available to run at a command line prompt, see the following in the Javadoc:

- Options for the JarMaker base class
- Extended options for the AS00ToolboxJarMaker subclass

Применение AS400ToolboxJarMaker

Класс AS400ToolboxJarMaker позволяет работать с файлами JAR несколькими способами:

- Распаковывать файлы, находящиеся в файлах JAR
- Разбивать большие файлы JAR на несколько файлов JAR меньшего размера
- Exclude any IBM Toolbox for Java files that your application does not need to run

Распаковка файла JAR

Предположим, что вы хотите распаковать один файл из архива JAR. Класс AS400ToolboxJarMaker позволяет получить файл из архива и поместить его в одно из следующих мест:

- Current directory (extract(jarFile))
- Another directory (extract(jarFile, outputDirectory))

Например, следующий фрагмент кода позволяет распаковать класс AS400.class и все зависящие от него классы из архива jt400.jar:

```
java utilities.AS400ToolboxJarMaker -source jt400.jar
    -extract outputDir
    -requiredFile com/ibm/as400/access/AS400.class
```

Разбиение одного файла JAR на несколько файлов JAR меньшего размера

Предположим, вы хотите разбить файл JAR на файлы меньшего размера, задав максимально допустимый размер файла JAR. AS400ToolboxJarMaker, accordingly, provides you with the split(jarFile, splitSize) function.

В следующем примере файл jt400.jar разбивается на файлы JAR размером не более 300 Кб:

```
java utilities.AS400ToolboxJarMaker -split 300
```

Удаление ненужных файлов из архива JAR

With AS400ToolboxJarMaker, you can exclude any IBM Toolbox for Java files not needed by your application by selecting only the IBM Toolbox for Java components, languages, and CCSIDs that you need to make your application run. Кроме того, AS400ToolboxJarMaker позволяет добавлять и удалять файлы JavaBean, связанные с выбранными компонентами.

For example, the following command creates a JAR file that contains only those IBM Toolbox for Java classes needed to make the CommandCall and ProgramCall components of the IBM Toolbox for Java work:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall
```

Кроме этого, если преобразование строк между кодировкой Unicode и набором двухбайтовых символов (DBCS) не требуется, вы можете сократить размер файла JAR на 400 Кб, опустив ненужные таблицы преобразования с помощью опции -ccsid:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall -ccsid 61952
```

Примечание: Классы преобразования не входят в состав классов вызова программ. Если в файл JAR добавляются классы вызова программ, классы преобразования должны быть указаны явно с помощью опции -ccsid.

JarMaker Javadoc

AS400ToolboxJarMaker Javadoc

Components supported by IBM Toolbox for Java:

Ниже приведены идентификаторы компонентов, которые можно указывать при вызове инструмента AS400ToolboxJarMaker.

- В столбце Компоненты приведены названия компонентов.
- В столбце Ключевое слово указаны ключевые слова, указываемые после опции `-component`.
- В столбце Константы перечислены значения типа Integer, указываемые в методах setComponents() и getComponents().

Компонент	Ключевое слово	Константа
Объект сервера	AS400	AS400ToolboxJarMaker.AS400
Вызов команды	CommandCall	AS400ToolboxJarMaker.COMMAND_CALL
Пул соединений	ConnectionPool	AS400ToolboxJarMaker.CONNECTION_POOL
Области данных	DataArea	AS400ToolboxJarMaker.DATA_AREA
Преобразование и описание данных	DataDescription	AS400ToolboxJarMaker.DATA_DESCRIPTION
Очереди данных	DataQueue	AS400ToolboxJarMaker.DATA_QUEUE
Цифровые сертификаты	DigitalCertificate	AS400ToolboxJarMaker.DIGITAL_CERTIFICATE
FTP	FTP	AS400ToolboxJarMaker.FTP
Интегрированная файловая система	IntegratedFileSystem	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM
JAAS	JAAS	AS400ToolboxJarMaker.JAAS

Компонент	Ключевое слово	Константа
Вызов приложения Java	JavaApplicationCall	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL
JDBC	JDBC	AS400ToolboxJarMaker.JDBC
Задания и очереди заданий	Job	AS400ToolboxJarMaker.JOB
Сообщения и очереди сообщений	Message	AS400ToolboxJarMaker.MESSAGE
Числовые типы данных	NumericDataTypes	AS400ToolboxJarMaker.NUMERIC_DATA_TYPES
NetServer	NetServer	AS400ToolboxJarMaker.NETSERVER
Сетевая печать	Print	AS400ToolboxJarMaker.PRINT
Вызов команды	ProgramCall	AS400ToolboxJarMaker.PROGRAM_CALL
Доступ на уровне записей	RecordLevelAccess	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS
Защищенный сервер	SecureAS400	AS400ToolboxJarMaker.SECURE_AS400
Вызов служебных программ	ServiceProgramCall	AS400ToolboxJarMaker.SERVICE_PROGRAM_CALL
Состояние системы	SystemStatus	AS400ToolboxJarMaker.SYSTEM_STATUS
Системные значения	SystemValue	AS400ToolboxJarMaker.SYSTEM_VALUE
Трассировка и регистрация	Trace	AS400ToolboxJarMaker.TRACE
Пользователи и группы	User	AS400ToolboxJarMaker.USER
Пользовательское пространство	UserSpace	AS400ToolboxJarMaker.USER_SPACE
Визуальный объект сервера	AS400Visual	AS400ToolboxJarMaker.AS400_VISUAL
Визуальный вызов команд	CommandCallVisual	AS400ToolboxJarMaker.COMMAND_CALL_VISUAL
Визуальные очереди данных	DataQueueVisual	AS400ToolboxJarMaker.DATA_QUEUE_VISUAL
Визуальная интегрированная файловая система	IntegratedFileSystemVisual	AS400ToolboxJarMaker.INTEGRATED_FILE_SYSTEM_VISUAL
Визуальный вызов приложения Java	JavaApplicationCallVisual	AS400ToolboxJarMaker.JAVA_APPLICATION_CALL_VISUAL
Визуальный JDBC	JDBCVisual	AS400ToolboxJarMaker.JDBC_VISUAL
Визуальные задания и очереди заданий	JobVisual	AS400ToolboxJarMaker.JOB_VISUAL
Визуальные сообщения и очереди сообщений	MessageVisual	AS400ToolboxJarMaker.MESSAGE_VISUAL
Визуальная сетевая печать	PrintVisual	AS400ToolboxJarMaker.PRINT_VISUAL

Компонент	Ключевое слово	Константа
Визуальный вызов программ	ProgramCallVisual	AS400ToolboxJarMaker.PROGRAM_CALL_VISUAL
Визуальный доступ на уровне записей	RecordLevelAccessVisual	AS400ToolboxJarMaker.RECORD_LEVEL_ACCESS_VISUAL
Визуальные пользователи и группы	UserVisual	AS400ToolboxJarMaker.USER_VISUAL

CCSID and encoding values supported by IBM Toolbox for Java:

IBM Toolbox for Java is shipped with a set of conversion tables, named according to the CCSID.

R These tables are used internally by IBM Toolbox for Java classes (such as CharConverter) when converting data that is
R transferred to or from a System i5. Например, таблица преобразования для CCSID 1027 находится в файле
R com/ibm/as400/access/ConvTable1027.class. Conversion tables for the following CCSIDs are included in the IBM
R Toolbox for Java jar file; other encodings are supported by using the JDK. Таблицы преобразования больше не
R загружаются с сервера. Все CCSID, для которых не может быть найдена таблица преобразования или
R кодировка JDK, вызовут исключительную ситуацию. Некоторые из перечисленных таблиц могут совпадать с
R таблицами, включенными в JDK. В настоящий момент IBM Toolbox for Java поддерживает 122 значения
R CCSID i5/OS.

R For additional information about CCSIDs, including a complete list of CCSIDs that are recognized on the System i
R platform, see Globalization.

Supported CCSIDs in IBM Toolbox for Java

CCSID	Формат	Описание
37	Однобайтовый EBCDIC	США и другие
273	Однобайтовый EBCDIC	Австрия, Германия
277	Однобайтовый EBCDIC	Дания, Норвегия
278	Однобайтовый EBCDIC	Финляндия, Швеция
280	Однобайтовый EBCDIC	Италия
284	Однобайтовый EBCDIC	Испания, Латинская Америка
285	Однобайтовый EBCDIC	Великобритания
290	Однобайтовый EBCDIC	Японский (катакана, только однобайтовый)
297	Однобайтовый EBCDIC	Франция
300	Двухбайтовый EBCDIC	Японский (графика, подмножество из 16684)
367	ASCII/ISO/Windows	ASCII (стандарт ANSI X3.4)
420	Однобайтовый EBCDIC (двунаправленный)	Арабский EBCDIC ST4
423	Однобайтовый EBCDIC	Греческий (для совместимости; см. 875)
424	Однобайтовый EBCDIC (двунаправленный)	Иврит EBCDIC ST4
437	ASCII/ISO/Windows	ASCII (USA PC)
500	Однобайтовый EBCDIC	Латиница-1 (MNCS)

CCSID	Формат	Описание
720	ASCII/ISO/Windows	Арабский (MS-DOS)
737	ASCII/ISO/Windows	Греческий (MS-DOS)
775	ASCII/ISO/Windows	Балтика (MS-DOS)
813	ASCII/ISO/Windows	ISO 8859-7 (Греческий/Латиница)
819	ASCII/ISO/Windows	ISO 8859-1 (Латиница-1)
833	Однобайтовый EBCDIC	Корейский (только однобайтовый)
834	Двухбайтовый EBCDIC	Корейский (графика, подмножество из 4930)
835	Двухбайтовый EBCDIC	Традиционный китайский (графика)
836	Однобайтовый EBCDIC	Упрощенный китайский (только однобайтовый)
837	Двухбайтовый EBCDIC	Упрощенный китайский (графика)
838	Однобайтовый EBCDIC	Тайский
850	ASCII/ISO/Windows	Латиница-1
851	ASCII/ISO/Windows	Греческий
852	ASCII/ISO/Windows	Латиница-2
855	ASCII/ISO/Windows	Кириллица
857	ASCII/ISO/Windows	Турецкий
860	ASCII/ISO/Windows	Португальский
861	ASCII/ISO/Windows	Исландия
862	ASCII/ISO/Windows (двунаправленный)	Иврит ASCII ST4
863	ASCII/ISO/Windows	Канада
864	ASCII/ISO/Windows (двунаправленный)	Арабский ASCII ST5
865	ASCII/ISO/Windows	Дания/Норвегия
866	ASCII/ISO/Windows	Кириллица/Русский
869	ASCII/ISO/Windows	Греческий
870	Однобайтовый EBCDIC	Латиница-2
871	Однобайтовый EBCDIC	Исландия
874	ASCII/ISO/Windows	Тайский (подмножество из 9066)
875	Однобайтовый EBCDIC	Греческий
878	ASCII/ISO/Windows	Русский
880	Однобайтовый EBCDIC	Кириллица (многоязычная; для совместимости; см. 1025)
912	ASCII/ISO/Windows	ISO 8859-2 (Латиница-2)
914	ASCII/ISO/Windows	ISO 8859-4 (Латиница-4)
915	ASCII/ISO/Windows	ISO 8859-5 (Кириллица, 8 разрядов)
916	ASCII/ISO/Windows (двунаправленный)	ISO 8859-8 (Иврит) ST5
920	ASCII/ISO/Windows	ISO 8859-9 (Латиница-5)
921	ASCII/ISO/Windows	ISO 8859-13 (Балтика; 8 разрядов)

CCSID	Формат	Описание
922	ASCII/ISO/Windows	Эстония ISO-8
923	ASCII/ISO/Windows	ISO 8859-15 (Латиница-9)
930	Смешанный EBCDIC	Японский (подмножество из 5026)
933	Смешанный EBCDIC	Корейский (подмножество из 1364)
935	Смешанный EBCDIC	Упрощенный китайский (подмножество из 1388)
937	Смешанный EBCDIC	Традиционный китайский
939	Смешанный EBCDIC	Японский (подмножество из 5035)
1025	Однобайтовый EBCDIC	Кириллица
1026	Однобайтовый EBCDIC	Турецкий
1027	Однобайтовый EBCDIC	Японский (латиница, только однобайтовый)
1046	ASCII/ISO/Windows (двунаправленный)	Windows, арабский ST5
1089	ASCII/ISO/Windows (двунаправленный)	ISO 8859-6 (арабский) ST5
1112	Однобайтовый EBCDIC	Балтика, многоязычный
1122	Однобайтовый EBCDIC	Эстонский
1123	Однобайтовый EBCDIC	Украина
1125	ASCII/ISO/Windows	Украина
1129	ASCII/ISO/Windows	Вьетнамский
1130	Однобайтовый EBCDIC	Вьетнамский
1131	ASCII/ISO/Windows	Беларусь
1132	Однобайтовый EBCDIC	Лаос
1140	Однобайтовый EBCDIC	США и другие (с поддержкой евро)
1141	Однобайтовый EBCDIC	Австрия, Германия (с поддержкой евро)
1142	Однобайтовый EBCDIC	Дания, Норвегия (с поддержкой евро)
1143	Однобайтовый EBCDIC	Финляндия, Швеция (с поддержкой евро)
1144	Однобайтовый EBCDIC	Италия (с поддержкой евро)
1145	Однобайтовый EBCDIC	Испания, Латинская Америка (с поддержкой евро)
1146	Однобайтовый EBCDIC	Великобритания (с поддержкой евро)
1147	Однобайтовый EBCDIC	Франция (с поддержкой евро)
1148	Однобайтовый EBCDIC	Латиница-1 (MNCS) (с поддержкой евро)
1149	Однобайтовый EBCDIC	Исландия (с поддержкой евро)
1200	Unicode	Unicode UCS-2 (в начале младший байт)
1250	ASCII/ISO/Windows	Windows Latin-2
1251	ASCII/ISO/Windows	Windows, кириллица
1252	ASCII/ISO/Windows	Windows Latin-1

CCSID	Формат	Описание
1253	ASCII/ISO/Windows	Windows, греческий
1254	ASCII/ISO/Windows	Windows, турецкий
1255	ASCII/ISO/Windows (двунаправленный)	Windows, иврит ST5
1256	ASCII/ISO/Windows (двунаправленный)	Windows, арабский ST5
1257	ASCII/ISO/Windows	Windows, балтийские
1258	ASCII/ISO/Windows	Windows, вьетнамский
1364	Смешанный EBCDIC	Японский
1388	Смешанный EBCDIC	Упрощенный китайский
1399	Смешанный EBCDIC	Японский (начиная с V4R5)
4396	Двухбайтовый EBCDIC	Японский (подмножество из 300)
4930	Двухбайтовый EBCDIC	Корейский
4931	Двухбайтовый EBCDIC	Традиционный китайский (подмножество из 835)
4933	Двухбайтовый EBCDIC	Упрощенный китайский (графика GBK)
4948	ASCII/ISO/Windows	Латиница-2 (подмножество из 852)
4951	ASCII/ISO/Windows	Кириллица (подмножество из 855)
5026	Смешанный EBCDIC	Японский
5035	Смешанный EBCDIC	Японский
5123	Однобайтовый EBCDIC	Японский (только однобайтовый, с поддержкой евро)
5351	ASCII/ISO/Windows (двунаправленный)	Windows, иврит (с поддержкой евро) ST5
8492	Двухбайтовый EBCDIC	Японский (подмножество из 300)
8612	Однобайтовый EBCDIC	Арабский EBCDIC ST5
9026	Двухбайтовый EBCDIC	Корейский (подмножество из 834)
9029	Двухбайтовый EBCDIC	Упрощенный китайский (подмножество из 4933)
9066	ASCII/ISO/Windows	Тайский (расширенный SBCS)
12588	Двухбайтовый EBCDIC	Японский (подмножество из 300)
13122	Двухбайтовый EBCDIC	Корейский (подмножество из 834)
16684	Двухбайтовый EBCDIC	Японский (в V4R5)
17218	Двухбайтовый EBCDIC	Корейский (подмножество из 834)
12708	Однобайтовый EBCDIC	Арабский EBCDIC ST7
13488	Unicode	Unicode UCS-2 (в начале старший байт)
28709	Однобайтовый EBCDIC	Традиционный китайский (только однобайтовый)
61952	Unicode	i5/OS Unicode (used primarily in the integrated file system)
62211	Однобайтовый EBCDIC	Иврит EBCDIC ST5
62224	Однобайтовый EBCDIC	Арабский EBCDIC ST6

CCSID	Формат	Описание
62235	Однобайтовый EBCDIC	Иврит EBCDIC ST6
62245	Однобайтовый EBCDIC	Иврит EBCDIC ST10

Класс CommandHelpRetriever

The CommandHelpRetriever class retrieves help text for i5/OS control language (CL) commands and generates that text either in HTML or User Interface Manager (UIM) format. Класс CommandHelpRetriever можно запускать с помощью командной строки или вызывать в программе на Java.

Для применения CommandHelpRetriever на сервере должна быть установлена система i5/OS версии V5R1 или выше, а в переменной CLASSPATH должны быть указаны анализатор XML и обработчик XSL. For more information, see “Анализатор XML и обработчик XSLT” на стр. 418.

Кроме того, класс CommandHelpRetriever применяется командой CL Создать документацию для команды (GENCMDDOC). Поэтому для применения функций класса CommandHelpRetriever можно воспользоваться командой GENCMDDOC. For more information, see Generate Command Documentation (GENCMDDOC) in the CL reference topic.

Запуск класса CommandHelpRetriever из командной строки

Класс CommandHelpRetriever можно запустить из командной строки в виде отдельной программы. Для этого необходимо указать следующий набор обязательных параметров:

R • The library on your server that contains the CL command. Команды системы размещены в библиотеке QSYS.
• Команду CL.

R You can also pass optional parameters to CommandHelpRetriever that include the server, the user ID, password, and
R the location for the generated file.

Дополнительная информация приведена в разделе документации Java для класса CommandHelpRetriever.

Пример: Запуск класса CommandHelpRetriever из командной строки

В следующем примере в текущем каталоге будет создан файл HTML CRTLIB.html.

Примечание: Пример команды приведен на двух строках для большей наглядности. Введите команду в одной строке.

```
     java com.ibm.as400.util.CommandHelpRetriever -library QSYS
-command CRTLIB
          -system ИмяСистемы -userid ИДпользователя -password Пароль
```

Добавление класса CommandHelpRetriever в программу

С помощью класса CommandHelpRetriever можно также просмотреть в программе на Java справку для выбранных команд CL. После создания объекта CommandHelpRetriever можно использовать методы generateHTML и generateUIM для создания справки в соответствующих форматах.

При применении generateHTML() созданный файл HTML можно просмотреть как в заданной, так и в отдельной группе панелей.

В приведенном ниже примере создается объект CommandHelpRetriever и объекты String, представляющие справку для команды CRTLIB в форматах HTML и UIM.

```
CommandHelpRetriever helpGenerator = new CommandHelpRetriever();
AS400 system = new AS400("ИмяСистемы", "ИДпользователя", "Пароль");
Command crtlibCommand = new Command(system, "/QSYS.LIB/CRTLIB.CMD");
String html = helpGenerator.generateHTML(crtlibCommand);
String uim = helpGenerator.generateUIM(crtlibCommand);
```

Информация, связанная с данной

CommandHelpRetriever Javadoc

Класс CommandPrompter

The IBM Toolbox for Java CommandPrompter class prompts for the parameter on a given command.

The CommandPrompter classoffers functionality that is similar to the CL command prompt (pressing F4) and the same as the Management Central command prompt.

Для применения класса CommandPrompter необходимо, чтобы на сервере была установлена i5/OS версии V4R4 или выше. For more information, see System i Navigator Information APARs and view the Required Fixes for Graphical Command Prompter Support.

Кроме того, для работы с CommandPrompter необходимо поместить в каталог CLASSPATH следующие jar-файлы:

- jt400.jar
- jui400.jar
- util400.jar
- jhall.jar

В параметре CLASSPATH должен быть также указан анализатор XML. Дополнительная информация о работе с соответствующим анализатором XML приведена на следующей странице:

“Анализатор XML и обработчик XSLT” на стр. 418

Все эти файлы jar, кроме jhall.jar, входят в состав IBM Toolbox for Java. For more information about IBM Toolbox for Java jar files, see Jar files. For more information about downloading jhall.jar, see the Sun JavaHelp Web site .

Для создания объекта CommandPrompter вы должны задать его параметры для родительского фрейма, запускающего приглашение, объект AS400, в котором будет выдано приглашение команды, и текст команды. В качестве текста команды можно указать имя команды, полную строку команды или часть имени команды, например `crt*`.

Меню CommandPrompter - это модальное окно диалога; вы должны закрыть его, прежде чем сможете вернуться к родительскому фрейму. CommandPrompter обрабатывает все ошибки, обнаруженные во время выдачи приглашения. Пример программы, демонстрирующий один из способов работы с CommandPrompter, приведен на следующей странице:

“Пример: Применение CommandPrompter” на стр. 723

Класс RunJavaApplication

The RunJavaApplication and VRunJavaApplication classes are utilities to run Java programs on the i5/OS JVM.

Unlike JavaApplicationCall and VJavaApplicationCall classes that you call from your Java program, RunJavaApplication and VRunJavaApplication are complete programs.

Класс RunJavaApplication реализован в виде утилиты командной строки. Он позволяет задавать переменные среды (например, CLASSPATH) для программ на Java. В качестве параметров указываются имя программы

на Java, которую нужно запустить, и ее параметры. После запуска программы на Java она может получать входные данные через стандартный ввод. Результаты выполняемой программы на Java направляются в стандартный вывод и стандартный файл ошибок.

Утилита VRunJavaApplication обладает теми же возможностями. В отличие от класса VJavaApplicationCall, в котором применяется графический пользовательский интерфейс, в классе JavaApplicationCall применяется интерфейс командной строки.

RunJavaApplication Javadoc

VRunJavaApplication Javadoc

“JavaApplicationCall” на стр. 60
Класс JavaApplicationCall позволяет клиенту запускать программы на Java, расположенные на сервере, с помощью сервера JVM.

“Класс VJavaApplicationCall” на стр. 266
The VJavaApplicationCall class allows you to run a Java application on the server from a client by using a graphical user interface (GUI).

Класс JPing

The JPing class is a command line utility that allows you to query your servers to see which services are running and which ports are in service. To query your servers from within a Java application, use the AS400JPing class.

See the JPing Javadoc for more information about using JPing from within your Java application.

Формат вызова JPing из командной строки:

```
java utilities.JPing система [опции]
```

где:

R • System = the system that you want to query
• [опции] - одна или несколько опций

Опции

Может быть указана одна или несколько из следующих опций. Сокращения опций приведены в круглых скобках.

-help (-h или -?)
Показывает текст справки.

-service *Служба_i5/OS* **(-s** *Служба_i5/OS***)**
Указывает службу, которой должен быть отправлен запрос. По умолчанию опрашиваются все службы. В этой опции можно задать следующие службы: as-file, as-netprt, as-rmtcmd, as-dtaq, as-database, as-ddm, as-central и as-signon.

-ssl Определяет, будут ли опрошены порты SSL. По умолчанию эти порты не опрашиваются.

-timeout (-t)
Задает значение тайм-аута в миллисекундах. Значение по умолчанию - 20000 или 20 секунд.

Пример: Вызов JPing из командной строки

Ниже приведен пример вызова команды для опроса службы as-dtaq (в том числе портов SSL) с тайм-аутом 5 секунд:

```
java utilities.JPing myServer -s as-dtaq -ssl -t 5000
```

Информация, связанная с данной

JPing Javadoc

AS400JPing Javadoc

Классы Vaccess

Пакет Vaccess и его классы устарели. Рекомендуется использовать пакет Access в сочетании с Java Swing.

IBM Toolbox for Java provides a set of graphical user interface (GUI) classes in the vaccess package. Эти классы являются классами доступа и предназначены для получения данных и представления их в удобном пользователю виде.

Java programs that use the IBM Toolbox for Java vaccess classes require the Swing package, which comes with the Java 2 Platform, Standard Edition (J2SE). For more information about Swing, see the Sun Java Foundation Classes

 Web site.

For more information about the relationships between theIBM Toolbox for Java GUI classes, the Access classes, and Java Swing, see the Vaccess classes diagram.

Примечание: AS400 panes are used with other vaccess classes (see items marked above with an asterisk) to present and allow manipulation of system resources.

When programming with the IBM Toolbox for Java graphical user interface components, use the Error events classes to report and handle error events to the user.

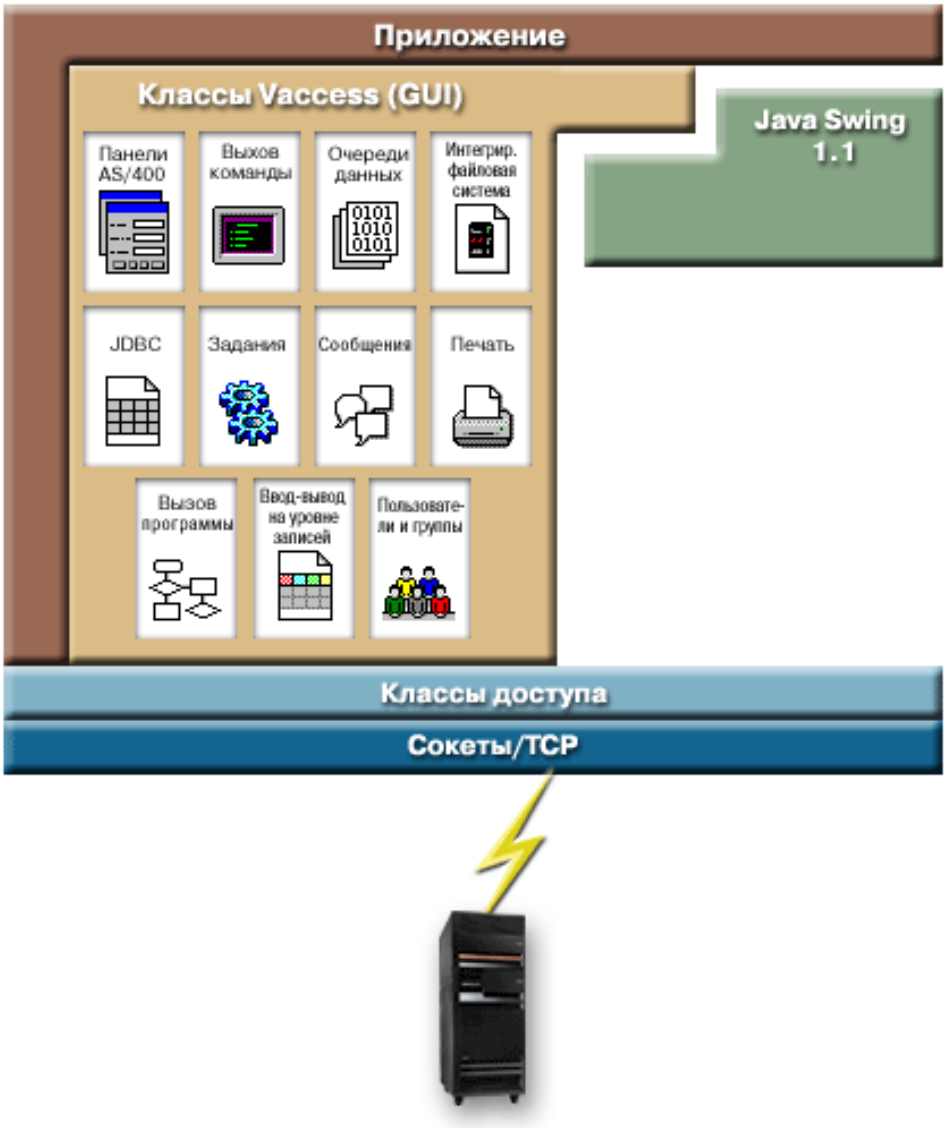
See Access classes for more information about accessing system data.

Классы Vaccess

Пакет Vaccess и его классы устарели. Рекомендуется использовать пакет Access в сочетании с Java Swing.

IBM Toolbox for Java предоставляет классы GUI для получения, представления и, в некоторых случаях, обработки данных сервера. Эти классы работают на базе Java Swing 1.1. На рис. 1 показана взаимосвязь между этими классами.

Рисунок 1: классы Vaccess



"Подробное описание рисунка 1: Классы Vaccess (rzahh508.gif)"

Подробное описание рисунка 1: Классы Vaccess (rzahh508.gif):

found in IBM Toolbox for Java: Vaccess package diagram

Данный рисунок иллюстрирует связь между классами пакета vaccess, пакетом доступа и классами Swing Java.

Описание

Рисунок состоит из следующих компонентов:

- Толстую коричневую рамку слева и сверху с именем 'Приложение', представляющую приложение на Java. Эта рамка приложения на Java ограничивает следующие изображения неправильной формы, соответствующие друг другу как части мозаики:
- Желтый многоугольник, представляющий классы графического интерфейса (GUI) IBM Toolbox for Java. Эта фигура содержит изображения функций класса vaccess меньшего размера.
- Зеленый многоугольник, представляющий классы Swing Java

- Расположенный ниже голубой прямоугольник с именем Классы доступа, представляющий классы пакета access IBM Toolbox for Java.
- Расположенный под голубым прямоугольником синий прямоугольник такого же размера с именем 'Сокеты/TCP.'

R • The bottom image is a picture of a System i.

R • A lightning bolt, which represents a socket connection from the Java application to the server, extends downward
R from Sockets/TCP (the dark blue bar) and connects to the server (the image of a System i).

Приложение на Java (область, ограниченная толстой коричневой рамкой) содержит классы vaccess (желтый многоугольник) и классы Swing Java (зеленый многоугольник). Классы vaccess позволяют приложению на Java получать доступ к следующим данным и функциям сервера (маленькие изображения внутри желтого многоугольника):

AS400Panes, CommandCall, DataQueues, Интегрированная файловая система, JDBC, задания, сообщения, печать, ProgramCall, ввод и вывод на уровне записей, пользователи и группы

R Приложение на Java применяет классы доступа IBM Toolbox for Java (голубой прямоугольник) для создания
R соединений между сокетами (синий прямоугольник). The socket connections enable the Java application to
R communicate (the lightning bolt) with the server (bottom image of the System i).

Класс AS400Panes

Объекты AS400Pane - это компоненты пакета Vaccess, предназначенные для работы с ресурсами сервера с помощью GUI. Для каждого типа ресурсов предусмотрен свой набор допустимых операций.

Все панели расширяют класс Component Java, поэтому их можно добавлять к любым фреймам, окнам и контейнерам AWT.

Предусмотрены следующие объекты AS400Pane:

- AS400DetailsPane presents a list of server resources in a table where each row displays various details about a single resource. Пользователь может выбрать в таблице произвольное число ресурсов.
- AS400ExplorerPane combines an AS400TreePane and AS400DetailsPane so that the resource selected in the tree is presented in the details.
- AS400JDBCDataSourcePane presents the property values of an AS400JDBCDataSource object.
- AS400ListPane presents a list of server resources and allows selection of one or more resources.
- AS400TreePane presents a tree hierarchy of server resources and allows selection of one or more resources.

Ресурсы сервера

Ресурсы сервера изображаются в GUI в виде значков с текстом. Они связаны иерархическими отношениями - у каждого ресурса может быть один родительский ресурс и произвольное число дочерних. Эти отношения предопределены; на их основе происходит отбор ресурсов, которые будут показаны в объекте AS400Pane. Например, объект VJobList может быть родительским для произвольного числа объектов VJob, и эти отношения будут графически отражены в объекте AS400Pane.

С помощью IBM Toolbox for Java можно работать со следующими ресурсами сервера:

- Объект VIFSDirectory представляет каталог IFS.
- Объекты VJob и VJobList представляют задание и список заданий.
- Объекты VMessageList и VMessageQueue представляют список сообщений, выданных объектом CommandCall или ProgramCall, и очередь сообщений.
- Объекты VPrinter, VPrinters и VPrinterOutput представляют принтер, список принтеров и список буферных файлов.
- Объект VUserList представляет список пользователей.

All resources are implementations of the VNode interface.

Указание корневого ресурса

Корневой ресурс объекта AS400Pane задается с помощью конструктора или метода setRoot(). Корневым называется ресурс, находящийся на верхнем уровне в иерархической структуре. Его применение зависит от конкретной панели:

- AS400ListPane presents all of the root's children in its list
- AS400DetailsPane presents all of the root's children in its table
- AS400TreePane uses the root as the root of its tree
- AS400ExplorerPane uses the root as the root of its tree

Допускаются любые сочетания панелей и корневых ресурсов.

Ниже приведен пример создания объекта AS400DetailsPane, представляющего список пользователей системы:

```
                    // Создание ресурса сервера для
                    // вывода списка пользователей.
                    // Предполагается, что объект AS400
                    // уже создан и инициализирован.
                    // инициализирована.
    VUserList userList = new VUserList (system);

                    // Создание объекта AS400DetailsPane
                    // и назначение списка пользователей
                    // в качестве корневого объекта.
    AS400DetailsPane detailsPane = new AS400DetailsPane ();
    detailsPane.setRoot (userList);

                    // Добавление панели со сведениями во фрейм.
                                        // Предполагается, что фрейм JFrame
                    // уже создан.
    frame.getContentPane ().add (detailsPane);
```

Загрузка содержимого

Сразу после создания объект AS400Pane и объекты ресурсов сервера находятся в стандартном начальном состоянии. Их содержимое не загружается из системы автоматически.

Для загрузки содержимого необходимо вызвать метод load(). В большинстве случаев на этом этапе устанавливается соединение с сервером для получения необходимой информации. Сбор информации может занять различное время, и в программе не всегда можно оценить его заранее. У вас есть несколько вариантов:

- Загрузка содержимого до добавления панели во фрейм. Фрейм не будет показан до того, как в него будет загружена вся информация.
- Загрузка содержимого после добавления панели во фрейм и вывода фрейма. Фрейм будет показан сразу, но в нем будет показана не вся информация. Курсор во фрейме примет форму, означающую, что система занята, а информация будет выводиться по мере загрузки.

Ниже приведен пример загрузки содержимого панели подробной информации перед добавлением его во фрейм:

```
                    // Загрузка содержимого фрейма.
                    // Предполагается, что панель
// detailsPane уже создана и
                    // инициализирована.
    detailsPane.load ();
```

```
                    // Добавление панели со сведениями во фрейм.
                                        // Предполагается, что фрейм JFrame
                    // уже создан.
    frame.getContentPane ().add (detailsPane);
```

Панели действий и свойств

Во время работы программы на Java пользователь может открыть всплывающее меню для любого ресурса сервера. Во всплывающем меню будет показан список возможных действий над ресурсом. Если пользователь выберет какое-либо действие в меню, оно будет выполнено. Для каждого типа ресурсов предусмотрен собственный набор действий.

В некоторых случаях во всплывающем меню предусмотрен пункт, позволяющий просмотреть свойства ресурса. В панели свойств указывается разнообразная информация о ресурсе, некоторые параметры которой иногда можно изменить.

С помощью метода setAllowActions() панели пользователь может указать, разрешено ли пользователю выполнять действия и просматривать свойства объекта.

Модели

Объекты AS400Pane реализованы по принципу ″модель-визуализация-управление″, согласно которому данные и пользовательский интерфейс разнесены в разные классы. В объектах AS400Pane интегрированы модели IBM Toolbox for Java и компоненты GUI Java . Описанные ниже модели могут применяться для управления ресурсами сервера. Компоненты Vaccess упрощают управление ресурсами, обеспечивая удобные средства взаимодействия пользователей с сервером.

Базовых функций объектов AS400Pane достаточно для выполнения большинства практических задач. Если же вам требуется более тонкое управление компонентом JFC, вы можете напрямую обратиться к модели сервера. Еще одно преимущество такого подхода заключается в том, что такие модели можно применять с разными компонентами Vaccess.

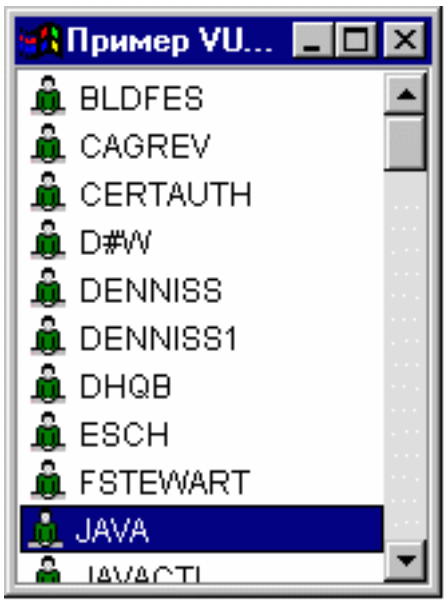
Предусмотрены следующие модели:

- AS400ListModel реализует интерфейс ListModel JFC в виде списка ресурсов сервера.Эта модель может применяться с объектом JList JFC.
- AS400DetailsModel реализует интерфейс TableModel JFC в виде таблицы ресурсов сервера, в каждой строке которой содержатся данные об одном ресурсе.Эта модель может применяться с объектом JTable JFC.
- AS400TreeModel реализует интерфейс TreeModel JFC в виде дерева ресурсов сервера.Она может применяться с объектом JTree JFC.

Примеры

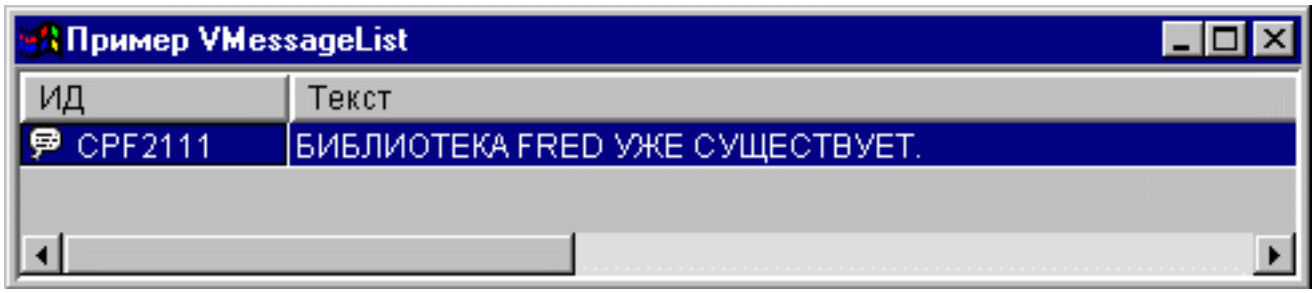
- Показывает список пользователей системы на панели AS400ListPane с помощью объекта VUserList. На рис. 1 показан окончательный результат:

  Рисунок 1: Применение панели AS400ListPane и объекта VUserList



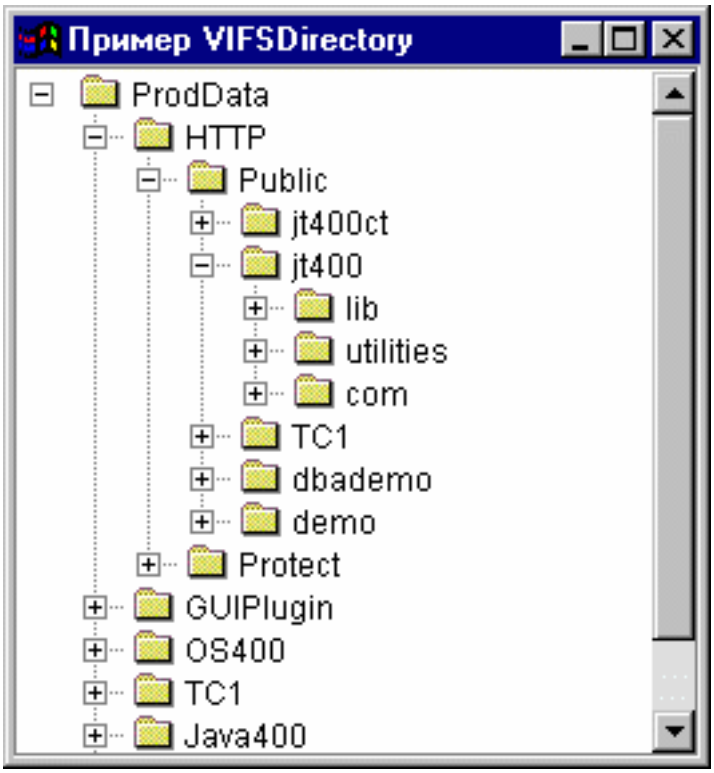
- Показывает список сообщений команды на панели AS400DetailsPane с помощью объекта VMessageList. На рис. 2 показан окончательный результат:

  Рисунок 2: Применение панели AS400DetailsPane и объекта VMessageList



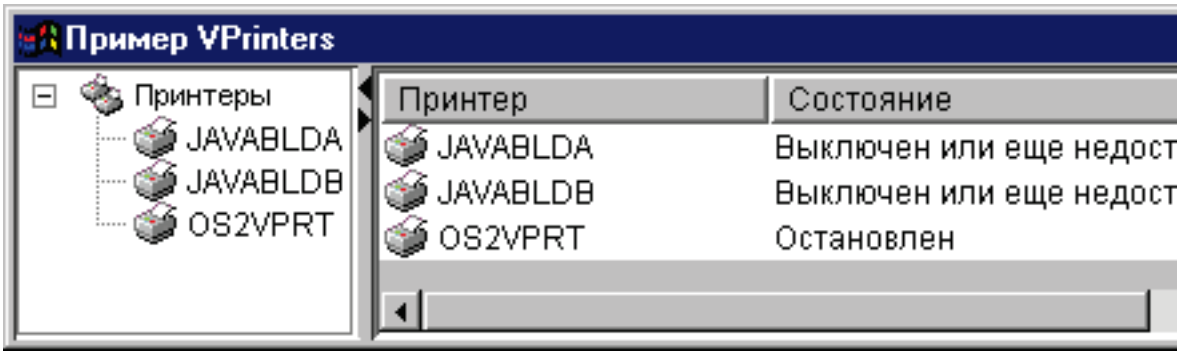
- Показывает дерево файлов интегрированной файловой системы на панели AS400TreePane с помощью объекта VIFSDirectory. На рис. 3 показан окончательный результат:

  Рисунок 3: Применение панели AS400TreePane и объекта VIFSDirectory



- Показывает ресурсы печати на панели AS400ExplorerPane с помощью объекта VPrinters. На рис. 4 показан окончательный результат:

Рисунок 4: Применение панели AS400ExplorerPane и объекта VPrinters



AS400DetailsPane Javadoc

AS400ExplorerPane Javadoc

AS400JDBCDataSourcePane Javadoc

AS400ListPane Javadoc

AS400TreePane Javadoc

VNode Javadoc

Вызов команды

С помощью компонентов GUI Vaccess, предназначенных для вызова команд, программы на Java могут создавать кнопки и меню для вызова неинтерактивных команд сервера.

Объект CommandCallButton представляет кнопку, при нажатии которой выполняется команда сервера.Класс CommandCallButton расширяет класс JButton из комплекта Java Foundation Classes (JFC), поэтому данная кнопка выглядит стандартно.

Аналогично, объект CommandCallMenuItem представляет меню, при выборе которого запускается команда сервера.Класс CommandCallMenuItem расширяет класс JMenuItem из комплекта JFC, поэтому все пункты меню выглядят стандартно.

Для работы с компонентами GUI вызова команд необходимо задать свойства system и command с помощью конструктора класса или методов setSystem() и setCommand().

В приведенном ниже примере создается объект CommandCallButton. Если при выполнении программы пользователь нажмет эту кнопку, будет создана библиотека ″FRED″.

```
                        // Создание объекта CommandCallButton.
                        // Предполагается, что объект AS400
// (system) уже создан и
                        // инициализирован.  На кнопке
                        // будет надпись "Нажми", но значка
                        // у кнопки не будет.
  CommandCallButton button = new CommandCallButton ("Нажми", null, system);

                        // Указание команды, которая будет выполняться при нажатии кнопки.
  button.setCommand ("CRTLIB FRED");

                        // Добавление кнопки во фрейм.
// Предполагается, что фрейм типа JFrame
                        // создается отдельно.
  frame.getContentPane ().add (button);
```

Во время выполнения команды сервера могут выдавать сообщения. Для определения момента запуска команды добавьте к кнопке или пункту меню обработчик ActionCompletedListener с помощью метода addActionCompletedListener().При запуске команды всем обработчикам будет передано событие ActionCompletedEvent.Обработчик событий может получать сообщения, выдаваемые командой сервера, с помощью метода getMessageList().

В этом примере создается объект ActionCompletedListener, обрабатывающий все сообщения сервера, созданные командой:

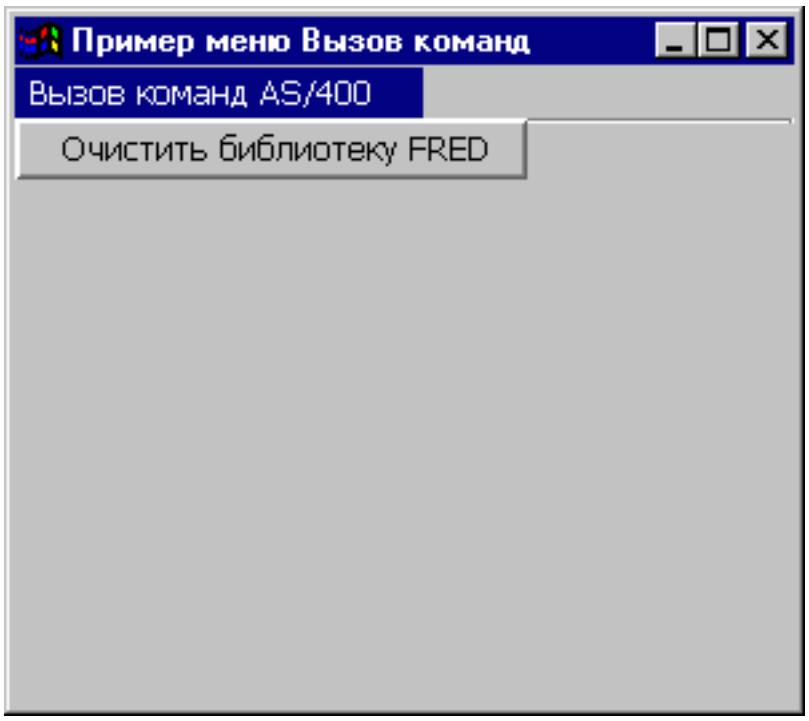
```
                        // Добавление обработчика ActionCompletedListener,
                        // реализованного с помощью анонимного
                        // внутреннего класса. Это наиболее удобный
// способ создания несложных обработчиков
                        // событий.
     button.addActionCompletedListener (new ActionCompletedListener ()
     {
          public void actionCompleted (ActionCompletedEvent event)
          {
                        // Отправка исходного кода события в
                        // CommandCallButton.
               CommandCallButton sourceButton = (CommandCallButton) event.getSource ();

                        // Получение списка сообщений, выданных
                        // командой.
               AS400Message[] messageList = sourceButton.getMessageList ();

                        // ... Обработка списка сообщений.
          }
     });
```

Примеры

В этом примере продемонстрировано применение объекта CommandCallMenuItem в приложении.

На рис. 1 показан компонент GUI CommandCall:

Рисунок 1: Компонент GUI CommandCall



Очереди данных

Компоненты GUI, связанные с очередями данных, позволяют программам на Java применять любые текстовые компоненты GUI Java Foundation Classes (JFC) для обмена информацией с очередями данных сервера.

The DataQueueDocument and KeyedDataQueueDocument classes are implementations of the JFC Document interface. Они могут напрямую использоваться с любыми текстовыми компонентами GUI JFC. В JFC предусмотрено несколько текстовых компонентов (например, однострочные (JTextField) и многострочные (JTextArea) области текста).

Документы очередей данных связывают содержимое текстовых компонентов с очередями данных сервера. (Текстовыми компонентами называются графические компоненты, предназначенные для вывода и редактирования текста). Программы на Java могут передавать данные из текстовых компонентов в очереди данных и обратно в любое время. Объект DataQueueDocument предназначен для работы с **последовательными**, а KeyedDataQueueDocument - с **ключевыми** очередями данных.

Для работы с объектом DataQueueDocument необходимо задать свойства system и path с помощью конструктора класса или методов setSystem() и setPath(). Затем нужно сделать DataQueueDocument корневым объектом текстового компонента с помощью конструктора компонента или метода setDocument(). Все вышесказанное в равной степени относится и к объекту KeyedDataQueueDocuments.

В приведенном ниже примере создается объект DataQueueDocument, содержимое которого связано с очередью данных:

```
                    // Создание объекта DataQueueDocument.
                    // Предполагается, что объект AS400
// (system) уже создан и
                    // инициализирован.
     DataQueueDocument dqDocument = new DataQueueDocument (system, "/QSYS.LIB/MYLIB.LIB/MYQUEUE.DTAQ");

                    // Создание области данных для
                    // документа.
     JTextArea textArea = new JTextArea (dqDocument);
```

```
                    // Добавление текстовой области во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
frame.getContentPane ().add (textArea);
```

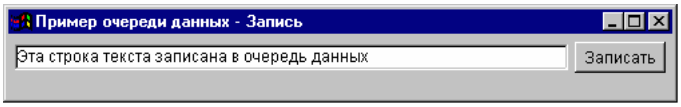
Первоначально текстовый компонент пуст. В него можно загрузить очередной элемент очереди данных с помощью метода read() или peek(). Метод write() позволяет записать содержимое текстового компонента в очередь данных. Учтите, что эти документы могут применяться только с элементами очередей данных типа String.

Примеры

Пример применения DataQueueDocument в приложении.

На рисунке 1 показан компонент GUI DataQueueDocument, применяемый с объектом JTextField. Предусмотрена специальная кнопка, нажав которую, пользователь может записать содержимое поля в очередь данных.

Рисунок 1: Компонент GUI DataQueueDocument



Информация, связанная с данной

DataQueueDocument Javadoc

KeyedDataQueueDocument Javadoc

События при ошибках

In most cases, the IBM Toolbox for Java GUI components fire error events instead of throw exceptions.

Событие при ошибке является внешним представлением исключительной ситуации, вызванной встроенным компонентом.

Вы можете создать обработчик событий для всех ошибок, о которых может сообщить определенный компонент GUI. При появлении исключительной ситуации обработчик будет вызываться для выполнения требуемой обработки. События при ошибках, напротив, по умолчанию игнорируются.

The IBM Toolbox for Java provides a graphical user interface component called ErrorDialogAdapter, which automatically displays a dialog to the user whenever an error event is fired.

Примеры

Ниже приведены примеры обработки ошибок и определения простого обработчика ошибок.

Пример: Обработка событий при ошибках с помощью окна диалога

Следующий пример иллюстрирует обработку события при возникновении ошибки с выводом окна диалога:

```
// Все компоненты графического интерфейса пользователя
// созданы и размечены. Теперь к компонентам нужно добавить обработчик ErrorDialogAdapter.
// Этот метод будет создавать сообщения об ошибках компонента с помощью
// окна диалога.

ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (parentFrame);
component.addErrorListener (errorHandler);
```

Пример: Определение обработчика ошибок

Вы можете создать обработчик событий по-другому. Для этого можно воспользоваться интерфейсом ErrorListener.

Ниже показан пример простого обработчика ошибок, который заносит сообщение об ошибке в System.out:

```
class MyErrorHandler
implements ErrorListener
{
   // Этот метод вызывается при возникновении ошибки.
   public void errorOccurred(ErrorEvent event)
   {
       Exception e = event.getException ();
       System.out.println ("Ошибка: " + e.getMessage ());
   }
}
```

Пример: Обработка событий при ошибках с помощью обработчика ошибок

В следующем примере продемонстрировано применение обработчика ошибок для компонента графического интерфейса:

```
MyErrorHandler errorHandler = new MyErrorHandler ();
component.addErrorListener (errorHandler);
```

Ссылки, связанные с данной

“Классы Vaccess” на стр. 251
Пакет Vaccess и его классы устарели. Рекомендуется использовать пакет Access в сочетании с Java Swing.

“Исключительные ситуации” на стр. 47
The IBM Toolbox for Java access classes throw exceptions when device errors, physical limitations, programming errors, or user input errors occur. Исключительные ситуации разделены на классы в соответствии с характером ошибок, а не по месту их возникновения.

Информация, связанная с данной

ErrorDialogAdapter Javadoc

IFS graphical user interface components

С помощью компонентов GUI, предназначенных для работы с IFS, программы на Java могут показывать каталоги и файлы IFS сервера в стандартных окнах GUI.

Для работы с компонентами GUI, связанными с IFS, необходимо задать свойства system и path с помощью конструктора класса или методов setDirectory() (для объекта IFSFileDialog) или setSystem() и setPath() (для объектов VIFSDirectory и IFSTextFileDocument).

Предусмотрены следующие компоненты:

Укажите другой путь вместо ″/QSYS.LIB″, поскольку этот каталог, как правило, содержит большой объем данных и их загрузка может занять много времени.

Класс IFSFileSystemView:

Этот класс устарел и заменен классом com.ibm.as400.access.IFSSystemView.

R The IFSFileSystemView provides a gateway to the System i5 integrated file system, for use when constructing
R javax.swing.JFileChooser objects.

JFileChooser - это стандартный инструмент Java для создания окон диалога, позволяющих искать и выбирать файлы; его рекомендуется применять вместо класса IFSFileDialog.

Пример: Применение класса IFSFileSystemView

Ниже приведен пример применения класса IFSFileSystemView.

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

// Работа с каталогом /Dir в системе myAS400.
AS400 system = new AS400("myAS400");
IFSJavaFile dir = new IFSJavaFile(system, "/Dir");
JFileChooser chooser = new JFileChooser(dir, new IFSFileSystemView(system));
Frame parent = new Frame();
int returnVal = chooser.showOpenDialog(parent);
if (returnVal == JFileChooser.APPROVE_OPTION) {
   IFSJavaFile chosenFile = (IFSJavaFile)(chooser.getSelectedFile());
   System.out.println("Вы выбрали файл " +
                      chosenFile.getName());
}
```

Информация, связанная с данной

IFSFileSystemView Javadoc

Класс VIFSFileDialog:

The IFSFileDialog class is a dialog that allows the user to traverse the directories of the integrated file system on the server and select a file. Названия кнопок этого окна можно задать по своему усмотрению. In addition, the caller can use FileFilter objects, which allow the user to limit the choices to certain files.

If the user selects a file in the dialog, use the getFileName() method to get the name of the selected file. Use the getAbsolutePath() method to get the full path name of the selected file.

В следующем примере создается окно диалога выбора файла с двумя фильтрами:

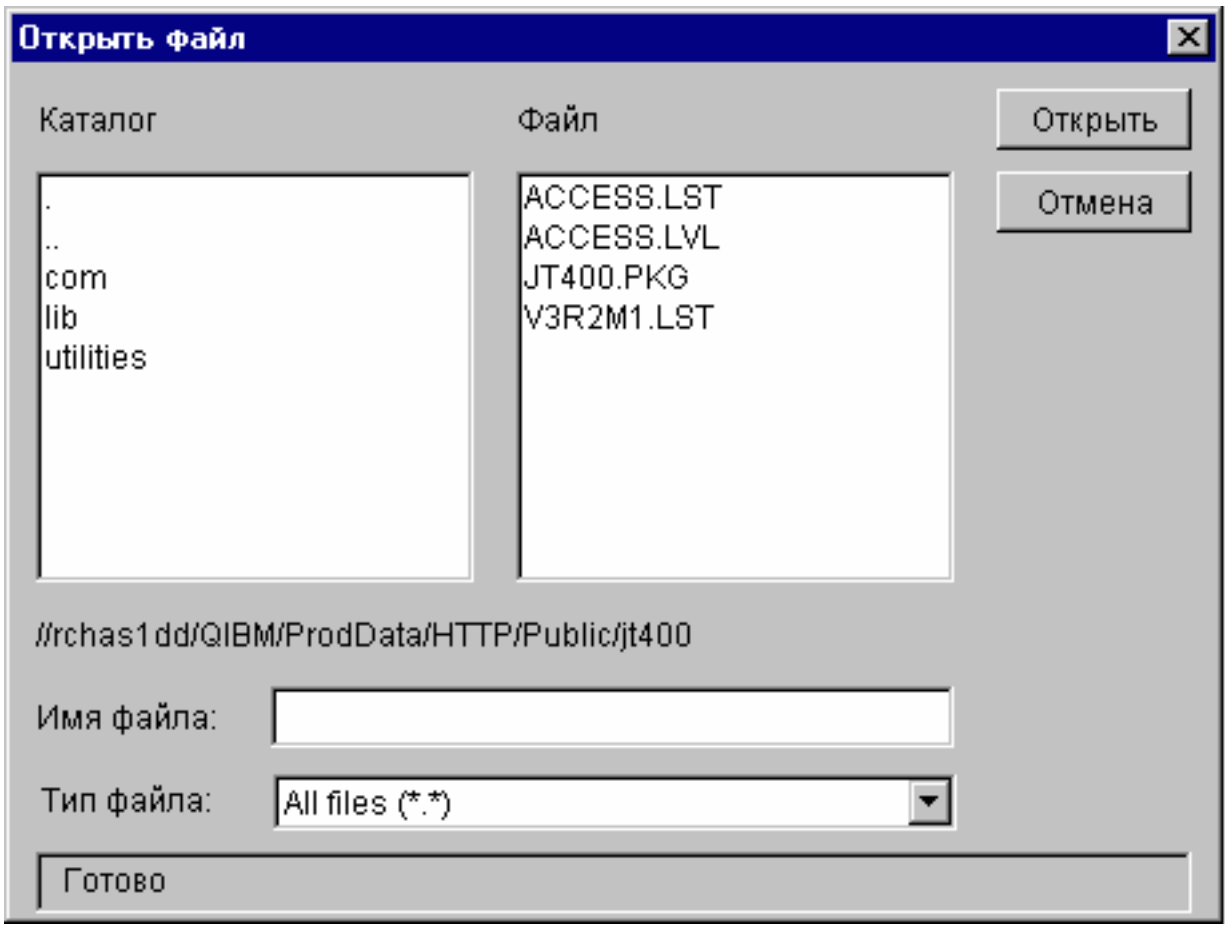
```
                  // Создание объекта IFSFileDialog
                  // с указанной строкой заголовка.
                  // Предполагается, что объект AS400
                  // и фрейм - это объекты JFrame,
                  // созданные и инициализированные отдельно.
IFSFileDialog dialog = new IFSFileDialog (frame, "Выберите файл", system);

                  // Указание списка фильтров для окна диалога.
                  // Первый фильтр будет применяться при первом
                  // просмотре окна диалога.
FileFilter[] filterList = {new FileFilter("Все файлы (*.*)", "*.*"),
                           new FileFilter("Файлы HTML (*.HTML)", "*.HTM")};
         // Применение фильтров к окну диалога.
dialog.setFileFilter (filterList, 0);

                  // Указание текста кнопок.
dialog.setOkButtonText("Открыть");
dialog.setCancelButtonText("Отмена");

                  // Показывается окно диалога. Если пользователь выбрал
                  // файл с помощью кнопки Открыть,
                  // будет напечатан путь к
                  // файлу.
if (dialog.showDialog () == IFSFileDialog.OK)
     System.out.println (dialog.getAbsolutePath ());
```

Пример

Показывает объект IFSFileDialog и выдает имя выбранного файла (если файл выбран).

На рис. 1 показан компонент GUI IFSFileDialog:

Рисунок 1: Компонент GUI IFSFileDialog



IFSFileDialog Javadoc

FileFilter Javadoc

Каталоги в объектах AS400Pane:

AS400Panes are GUI components that present and allow manipulation of one or more server resources. A VIFSDirectory object is a resource that represents a directory in the integrated file system for use in AS400Panes. Объекты AS400Pane и VIFSDirectory - это универсальный инструментарий для выполнения всевозможных операций над интегрированной файловой системой и ее каталогами и файлами.

Для работы с объектом VIFSDirectory необходимо задать свойства system и path с помощью конструктора класса или методов setSystem() и setPath(). Затем нужно сделать объект VIFSDirectory корневым объектом объекта AS400Pane с помощью конструктора или метода setRoot() объекта AS400Pane.

В объекте VIFSDirectory предусмотрены свойства, позволяющие определять наборы каталогов и файлов для представления в объектах AS400Pane. Метод setInclude() позволяет выбрать нужный тип объектов (каталоги, файлы или и то, и другое). С помощью метода setPattern() можно задать фильтр для имен файлов. В фильтре могут применяться символы подстановки ″*″ и ″?″. Similarly, use setFilter() to set a filter with an IFSFileFilter object.

Сразу после создания объекты AS400Pane и VIFSDirectory находятся в стандартном начальном состоянии. Информация о каталогах и файлах, хранящихся в корневом каталоге, не загружается из системы AS/400. Для загрузки информации нужно явно вызвать метод load() для соответствующего каталога.

Во время работы программы пользователь может выполнять действия над каталогами и файлами с помощью меню, появляющихся при нажатии правой кнопки мыши. В контекстном меню каталога могут быть предусмотрены следующие пункты:

- **Создать файл** - создание файла в каталоге. Файлу будет присвоено стандартное имя по умолчанию.
- **Создать каталог** - создание каталога со стандартным именем по умолчанию.
- **Переименовать** - изменение имени каталога.
- **Удалить** - удаление каталога.
- **Свойства** - просмотр свойств каталога, в частности, его расположения, количества файлов и подкаталогов и даты изменения.

В контекстном меню файла могут быть предусмотрены следующие пункты:

- **Правка** - изменение содержимого файла в отдельном окне.
- **Просмотр** - просмотр текстового файла в отдельном окне.
- **Переименовать** - изменение имени файла.
- **Удалить** - удаление файла.
- **Свойства** - просмотр свойств файла, в частности, его расположения, даты последнего изменения и атрибутов.

Пользователи могут работать только с теми каталогами и файлами, к которым у них есть права доступа. Кроме того, с помощью метода setAllowActions() можно запретить пользователям выполнять действия над объектами.

Следующий фрагмент кода создает объект VIFSDirectory и выводит его содержимое в панели AS400ExplorerPane:

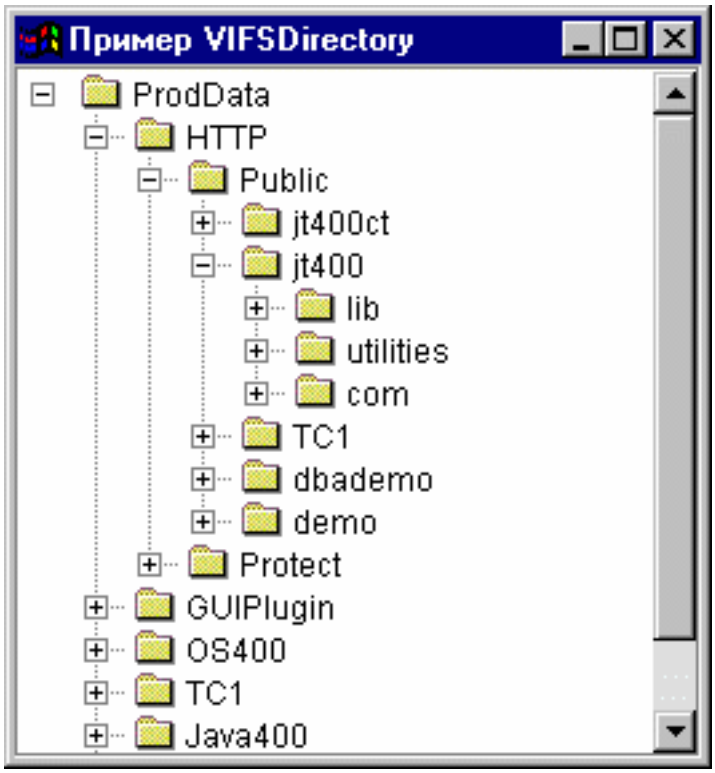
```
                    // Создание объекта VIFSDirectory.
                    // Предполагается, что "system" - это объект AS400, который
                    // уже создан и инициализирован.
                    //
VIFSDirectory root = new VIFSDirectory (system, "/DirectoryA/DirectoryB");

                    // Создание и загрузка объекта AS400ExplorerPane
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
explorerPane.load ();

                    // Добавление панели проводника во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
frame.getContentPane ().add (explorerPane);
```

Пример

Представляет иерархическое дерево объектов IFS в панели AS400TreePane с помощью объекта VIFSDirectory.

На рис. 1 показан компонент GUI VIFSDirectory:

Рисунок 1: Компонент GUI VIFSDirectory



“Класс AS400Panes” на стр. 253
Объекты AS400Pane - это компоненты пакета Vaccess, предназначенные для работы с ресурсами сервера с помощью GUI. Для каждого типа ресурсов предусмотрен свой набор допустимых операций.

VIFSDirectory Javadoc

IFSFileFilter Javadoc

Класс IFSTextFileDocument:

Документы текстовых файлов позволяют программам на Java применять любые текстовые компоненты GUI Java Foundation Classes (JFC) для работы с текстовыми файлами, хранящимися в IFS сервера. (Текстовыми компонентами называются графические компоненты, предназначенные для вывода и редактирования текста).

The IFSTextFileDocument class is an implementation of the JFC Document interface. Он может напрямую использоваться с любыми текстовыми компонентами GUI JFC. В JFC предусмотрено несколько текстовых компонентов (например, однострочные (JTextField) и многострочные (JTextArea) области текста).

Документы текстовых файлов связывают содержимое текстовых компонентов с текстовыми файлами. Программы на Java могут передавать данные из текстового компонента в текстовый файл и обратно в любое время.

Для работы с объектом IFSTextFileDocument необходимо задать свойства system и path с помощью конструктора класса или методов setSystem() и setPath(). После этого объект IFSTextFileDocument ″подключается″ к текстовому компоненту, как правило, с помощью конструктора этого компонента или метода setDocument().

Первоначально текстовый компонент пуст. С помощью метода load() в него можно загрузить данные из текстового файла. Метод save() позволяет записать содержимое текстового компонента в текстовый файл.

Ниже приведен пример создания и инициализации объекта IFSTextFileDocument:

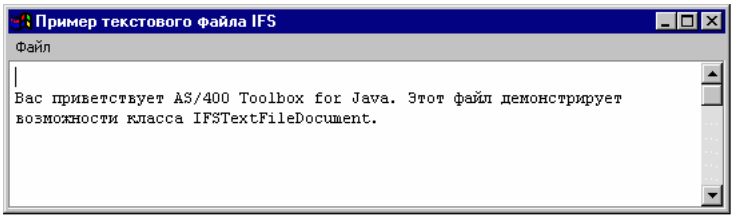
```
                    // Создание и загрузка объекта
                    // IFSTextFileDocument. Предполагается,
// Предполагается, что объект AS400
                    // уже создан и инициализирован.
     IFSTextFileDocument ifsDocument = new IFSTextFileDocument (system, "/DirectoryA/MyFile.txt");
     ifsDocument.load ();

                    // Создание области данных для
                    // документа.
     JTextArea textArea = new JTextArea (ifsDocument);

                    // Добавление текстовой области во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
     frame.getContentPane ().add (textArea);
```

Example

Представляет объект IFSTextFileDocument в панели JTextPane.

На рис. 1 показан компонент GUI IFSTextFileDocument:

Рисунок 1: Пример объекта IFSTextFileDocument



IFSTextFileDocument Javadoc

Класс VJavaApplicationCall

The VJavaApplicationCall class allows you to run a Java application on the server from a client by using a graphical user interface (GUI).

Графический интерфейс - это панель, состоящая из двух частей. В верхней части расположено окно вывода, в котором выдается информация, помещаемая программой на Java в стандартные потоки вывода и ошибок. В нижней части находится поле ввода, в котором пользователь задает переменные среды Java, указывает программу на Java и вводит информацию, передаваемую программе на Java в стандартном потоке ввода. Refer to the Java command options for more information.

С помощью этого примера можно создать следующий GUI для программы на Java.

Класс VJavaApplicationCall - это класс, вызываемый из программы на Java. However, the IBM Toolbox for Java also provides a utility that is a complete Java application that can be used to call your Java program from a workstation. Дополнительная информация приведена в разделе Класс RunJavaApplication.

Информация, связанная с данной

VJavaApplicationCall Javadoc

Классы JDBC

Компоненты GUI, относящиеся к JDBC, позволяют программам на Java выполнять различные операции над базами данных с помощью операторов и запросов языка SQL.

Предусмотрены следующие компоненты:

- SQLStatementButton и SQLStatementMenuItem - это кнопки и пункты меню, при активации которых запускаются операторы SQL.
- SQLStatementDocument - документ, который может применяться с любым текстовым компонентом GUI Java Foundation Classes (JFC) для выполнения оператора SQL.
- SQLResultSetFormPane - панель, представляющая результаты выполнения запроса SQL в виде формы.
- SQLResultSetTablePane - панель, представляющая результаты выполнения запроса SQL в виде таблицы.
- SQLResultSetTableModel - панель, предназначенная для работы с результатами запроса SQL в таблице.
- SQLQueryBuilderPane - панель, предназначенная для создания запросов SQL в интерактивном режиме.

Все компоненты GUI JDBC обращаются к базам данных с помощью драйвера JDBC. Для их работы необходимо, чтобы драйвер JDBC был зарегистрирован диспетчером драйверов JDBC. В приведенном ниже примере выполняется регистрация драйвера JDBC IBM Toolbox for Java:

```
                    // Регистрация драйвера JDBC.
 DriverManager.registerDriver (new
com.ibm.as400.access.AS400JDBCDriver ());
```

Соединения SQL

An SQLConnection object represents a connection to a database using JDBC. **Объект SQLConnection применяется практически всеми компонентами GUI JDBC.**

To use an SQLConnection, set the URL property using the constructor or setURL(). Это свойство указывает, с какой базой данных устанавливается соединение. Помимо этого, могут быть заданы следующие необязательные свойства:

- Use setProperties() to specify a set of JDBC connection properties.
- Use setUserName() to specify the user name for the connection.
- Use setPassword() to specify the password for the connection.

При создании объекта SQLConnection соединение с базой данных не устанавливается. Instead, it is made when getConnection() is called. Обычно этот метод автоматически вызывается компонентами GUI JDBC, но при необходимости его можно вызвать в любое время вручную.

Ниже приведен пример создания и инициализации объекта SQLConnection:

```
                    // Создание объекта SQLConnection.
    SQLConnection connection = new SQLConnection ();

                    // Указание URL и имени пользователя для соединения.
    connection.setURL ("jdbc:as400://MySystem");
    connection.setUserName ("Lisa");
```

Объект SQLConnection может одновременно применяться несколькими компонентами GUI JDBC. В этом случае все компоненты будут пользоваться одним соединением, что позволит повысить производительность и сократить объем занятых ресурсов. Однако можно поступить иначе, выделив каждому компоненту собственный объект SQL. Такая необходимость может возникнуть, например, для распределения операторов SQL по разным транзакциям.

When the connection is no longer needed, close the SQLConnection object using close(). Это позволит освободить ресурсы JDBC как на клиенте, так и на сервере.

SQLConnection Javadoc

Кнопки и пункты меню:

An SQLStatementButton object represents a button that issues an SQL (Structured Query Language) statement when pressed. Класс SQLStatementButton расширяет класс JButton из комплекта Java Foundation Classes (JFC), поэтому данная кнопка выглядит стандартно.

Similarly, an SQLStatementMenuItem object represents a menu item that issues an SQL statement when selected. Класс SQLStatementMenuItem расширяет класс JMenuItem из комплекта JFC, поэтому все пункты меню выглядят стандартно.

Для работы с этими классами нужно задать свойства connection и SQLStatement с помощью конструктора класса или методов setConnection() и setSQLStatement().

Ниже приведен пример создания объекта SQLStatementButton. Нажатие кнопки удаляет все записи из таблицы:

```
                        // Создание объекта SQLStatementButton.
                        // На кнопке будет написано "Удалить все",
                        // у нее не будет значка.
     SQLStatementButton button = new SQLStatementButton ("Удалить все");

                        // Указание свойств connection и SQLStatement.
                        // Предполагается, что объект типа
// "connection" - это объект SQLConnection,
                        // который уже создан и инициализирован.
     button.setConnection (connection);
     button.setSQLStatement ("DELETE FROM MYTABLE");

                        // Добавление кнопки во фрейм.
// Предполагается, что фрейм типа JFrame
                        // создается отдельно.
     frame.getContentPane ().add (button);
```

После выполнения оператора SQL его результаты можно получить с помощью методов getResultSet(), getMoreResults(), getUpdateCount() и getWarnings().

SQLStatementButton Javadoc

SQLStatementMenuItem Javadoc

Класс SQLStatementDocument:

The SQLStatementDocument class is an implementation of the Java Foundation Classes (JFC) Document interface. Он может напрямую использоваться с любыми текстовыми компонентами GUI JFC.

В JFC предусмотрено несколько текстовых компонентов (например, однострочные (JTextField) и многострочные (JTextArea) области текста). Объекты SQLStatementDocument связывают содержимое текстовых компонентов с объектами SQLConnection. Программа на Java может в любое время выполнять операторы SQL, указанные в содержимом документов, и обрабатывать их результаты.

Для работы с объектом SQLStatementDocument нужно задать свойство connection с помощью конструктора класса или метода setConnection(). После этого объект SQLStatementDocument ″подключается″ к текстовому компоненту, как правило, с помощью конструктора этого компонента или метода setDocument(). Use the execute() method at any time to run the SQL statement contained in the document.

Ниже приведен пример создания объекта SQLStatementDocument в компоненте GUI JTextField:

```
                        // Создание объекта SQLStatementDocument.
                        // Предполагается, что объект типа
// "connection" - это объект SQLConnection,
                        // который уже создан и инициализирован.
                        // В данном примере документ
                        // инициализируется стандартным запросом.
     SQLStatementDocument document = new SQLStatementDocument (connection, "SELECT * FROM QIWS.QCUSTCDT");

                        // Создание текстового поля
                        // документа.
     JTextField textField = new JTextField ();
     textField.setDocument (document);
```

```
                    // Добавление текстового поля во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
    frame.getContentPane ().add (textField);

                    // Выполнение оператора SQL, указанного в
                    // текстовом поле.
    document.execute ();
```

После выполнения оператора SQL его результаты можно получить с помощью методов getResultSet(), getMoreResults(), getUpdateCount() и getWarnings().

Информация, связанная с данной

SQLStatementDocument Javadoc

Класс SQLResultSetFormPane:

An SQLResultSetFormPane presents the results of an SQL (Structured Query Language) query in a form. В каждый момент времени в форме показана одна запись; пользователь может перемещаться по записям с помощью кнопок.

Для работы с объектом SQLResultSetFormPane нужно задать свойства connection и query. Set these properties by using the constructor or the setConnection() and setQuery() methods. Use load() to execute the query and present the first record in the result set. When the results are no longer needed, call close() to ensure that the result set is closed.

Следующий фрагмент кода создает объект SQLResultSetFormPane и добавляет его во фрейм:

```
                    // Создание объекта SQLResultSetFormPane.
                    // Предполагается, что объект типа
// "connection" - это объект SQLConnection,
                    // который уже создан и инициализирован.
    SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection, "SELECT * FROM QIWS.QCUSTCDT");
// Загрузка результатов.
    formPane.load ();

                    // Добавление панели с формой во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
    frame.getContentPane ().add (formPane);
```

Информация, связанная с данной

SQLResultSetFormPane Javadoc

Класс SQLResultSetTablePane:

An SQLResultSetTablePane presents the results of an SQL (Structured Query Language) query in a table. В каждой строке таблицы показана одна запись набора результатов, причем каждый столбец строки соответствует одному полю.

Для работы с объектом SQLResultSetTablePane нужно задать свойства connection и query. Set properties by using the constructor or the setConnection() and setQuery() methods. Use load() to execute the query and present the results in the table. When the results are no longer needed, call close() to ensure that the result set is closed.

Следующий фрагмент кода создает объект SQLResultSetTablePane и добавляет его во фрейм:

```
                    // Создание объекта SQLResultSetTablePane.
                    // Предполагается, что объект типа
// "connection" - это объект SQLConnection,
                    // который уже создан и инициализирован.
    SQLResultSetTablePane tablePane = new SQLResultSetTablePane (connection,
                                      "SELECT * FROM QIWS.QCUSTCDT");
                    // Загрузка результатов.
    tablePane.load ();
```

```
                    // Добавление панели с таблицей во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
    frame.getContentPane ().add (tablePane);
```

Пример

В данном примере показана панель SQLResultSetTablePane с таблицей результатов обработки запроса. В данном примере применяются следующие объекты: SQLStatementDocument (на рисунке помечен текстом ″Введите оператор SQL″) - документ, позволяющий задать произвольный оператор SQL, и SQLStatementButton (помечен текстом ″Удалить все строки″) - кнопка, при нажатии которой выполняется заранее оговоренное действие (удаляются все строки таблицы).

На рис. 1 показан компонент GUI SQLResultSetTablePane:

Рисунок 1: Компонент GUI SQLResultSetTablePane

Информация, связанная с данной

SQLResultSetTablePane Javadoc

Класс SQLResultSetTableModel:

Объект SQLResultSetTablePane реализован по принципу ″модель-визуализация-управление″, согласно которому данные и пользовательский интерфейс разнесены в разные классы. The implementation integrates SQLResultSetTableModel with the Java Foundation Classes' (JFC) JTable. Класс SQLResultSetTableModel обеспечивает управление результатами запросов, а объект JTable - визуализацию результатов и взаимодействие с пользователем.

Средств объекта SQLResultSetTablePane достаточно для выполнения большинства практических задач. Если же вам требуется более тонкое управление компонентом JFC, воспользуйтесь объектом SQLResultSetTableModel напрямую. Еще одно преимущество заключается в том, что объект SQLResultSetTableModel можно применять с разными компонентами GUI.

To use an SQLResultSetTableModel, set theconnection and query properties. Set these properties by using the constructor or the setConnection() and setQuery() methods. Use load() to execute the query and load the results. When the results are no longer needed, call close() to ensure that the result set is closed.

Следующий фрагмент кода создает объект SQLResultSetTableModel и выводит его в таблице JTable:

```
                    // Создание объекта SQLResultSetTableModel.
                    // Предполагается, что объект типа
// "connection" - это объект SQLConnection,
                    // который уже создан и инициализирован.
     SQLResultSetTableModel tableModel = new SQLResultSetTableModel (connection,
                                         "SELECT * FROM QIWS.QCUSTCDT");
                    // Загрузка результатов.
     tableModel.load ();

                    // Создание объекта JTable для модели.
     JTable table = new JTable (tableModel);

                    // Добавление таблицы во фрейм.
// Предполагается, что фрейм типа JFrame
                    // создается отдельно.
     frame.getContentPane ().add (table);
```

Информация, связанная с данной

SQLResultSetTableModel Javadoc

Редактор запросов SQL:

An SQLQueryBuilderPane presents an interactive tool for dynamically building SQL queries.

Для работы с SQLQueryPane нужно задать свойство connection. This property can be set using the constructor or the setConnection() method. Use load() to load data needed for the query builder graphical user interface. Use getQuery() to get the SQL query that the user has built.

Следующий фрагмент кода создает объект SQLQueryBuilderPane и добавляет его во фрейм:

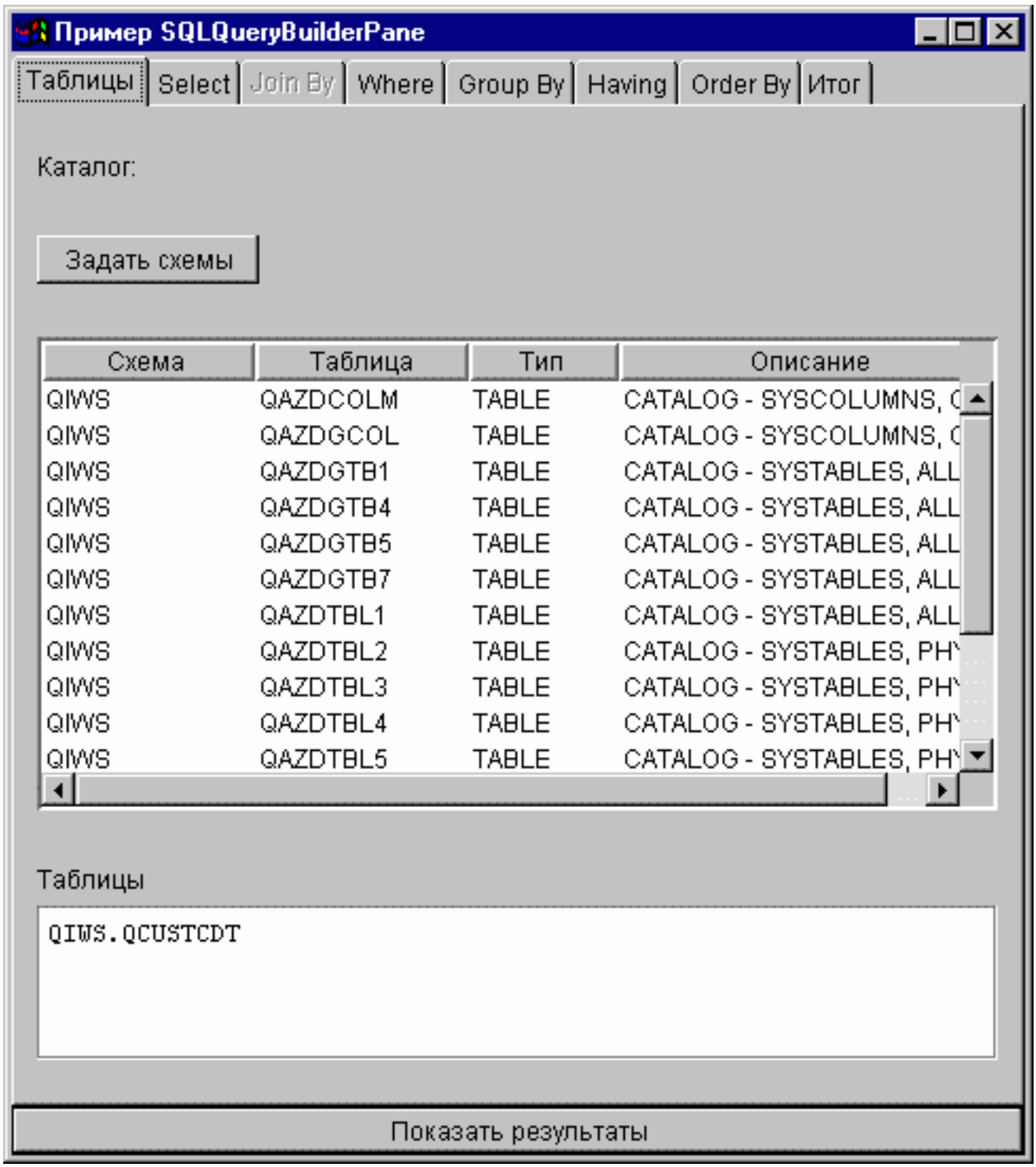
```
                    // Создание объекта SQLQueryBuilderPane.
                    // Предполагается, что объект типа
// "connection" - это объект SQLConnection,
                    // который уже создан и инициализирован.
     SQLQueryBuilderPane queryBuilder = new SQLQueryBuilderPane (connection);

                    // Загрузка данных, необходимых для
                    // создания запроса.
     queryBuilder.load ();

                    // Добавление панели во фрейм.
                    // Предполагается, что фрейм типа JFrame
// (frame) уже создан.
     frame.getContentPane ().add (queryBuilder);
```

Пример

Present an SQLQueryBuilderPane and a button. При нажатии кнопки результаты выполнения запроса выдаются в панели SQLResultSetFormPane в другом фрейме.

На рис. 1 показан компонент GUI SQLQueryBuilderPane:

Рисунок 1: Компонент GUI SQLQueryBuilderPane



Информация, связанная с данной

SQLQueryBuilderPane Javadoc

Задания

С помощью компонентов GUI Задания Vaccess программы на Java могут выводить списки заданий и сообщения из протоколов заданий сервера в окнах GUI.

Предусмотрены следующие компоненты:

- A VJobList object is a resource that represents a list of server jobs for use in AS400Panes.

- A VJob object is a resource that represents the list of messages in a job log for use in AS400Panes.

С помощью панелей AS400Panes и объектов VJobList и VJob можно создавать различные представления списков заданий и протоколов заданий.

Для работы с объектом VJobList нужно задать свойства system, name, number и user. Set these properties by using a constructor or through the setSystem(), setName(), setNumber(), and setUser() properties.

Для применения объекта VJob необходимо задать свойство system. Set this property by using a constructor or through the setSystem() method.

Объект VJobList или VJob добавляется в качестве корневого объекта панели AS400Pane с помощью конструктора панели или метода setRoot().

В объекте VJobList предусмотрены некоторые свойства, позволяющие задать критерии для отбора заданий, которые будут выведены на панели AS400Panes. Use setName() to specify that only jobs with a certain name appear. Use setNumber() to specify that only jobs with a certain number appear. Similarly, use setUser() to specify that only jobs for a certain user appear.

При создании объекты AS400Pane, VJobList и VJob инициализируются значениями по умолчанию. В них не загружаются списки заданий и сообщения из протоколов заданий. Для загрузки данных в объект нужно вызвать метод load() для этого объекта. В результате содержимое списка будет загружено из сервера.

При щелчке правой кнопкой мыши на имени задания, списке заданий или протоколе задания появляется контекстное меню. Выберите пункт **Свойства** в меню ярлыков и выполните следующие действия с выбранным объектом:

- Задание - Можно просмотреть свойства объекта (например, тип и состояние). Кроме того, можно изменить некоторые свойства.
- Список заданий - Можно просмотреть свойства объекта, в том числе имя, номер и имя пользователя. Можно также изменить содержимое списка.
- Сообщение из протокола задания - Можно просмотреть свойства объекта, в том числе полный текст сообщения, уровень серьезности и время создания.

Пользователи могут работать только с теми заданиями, к которым у них есть права доступа. Более того, с помощью метода setAllowActions() программа на Java может ограничить диапазон действий, разрешенных пользователям.

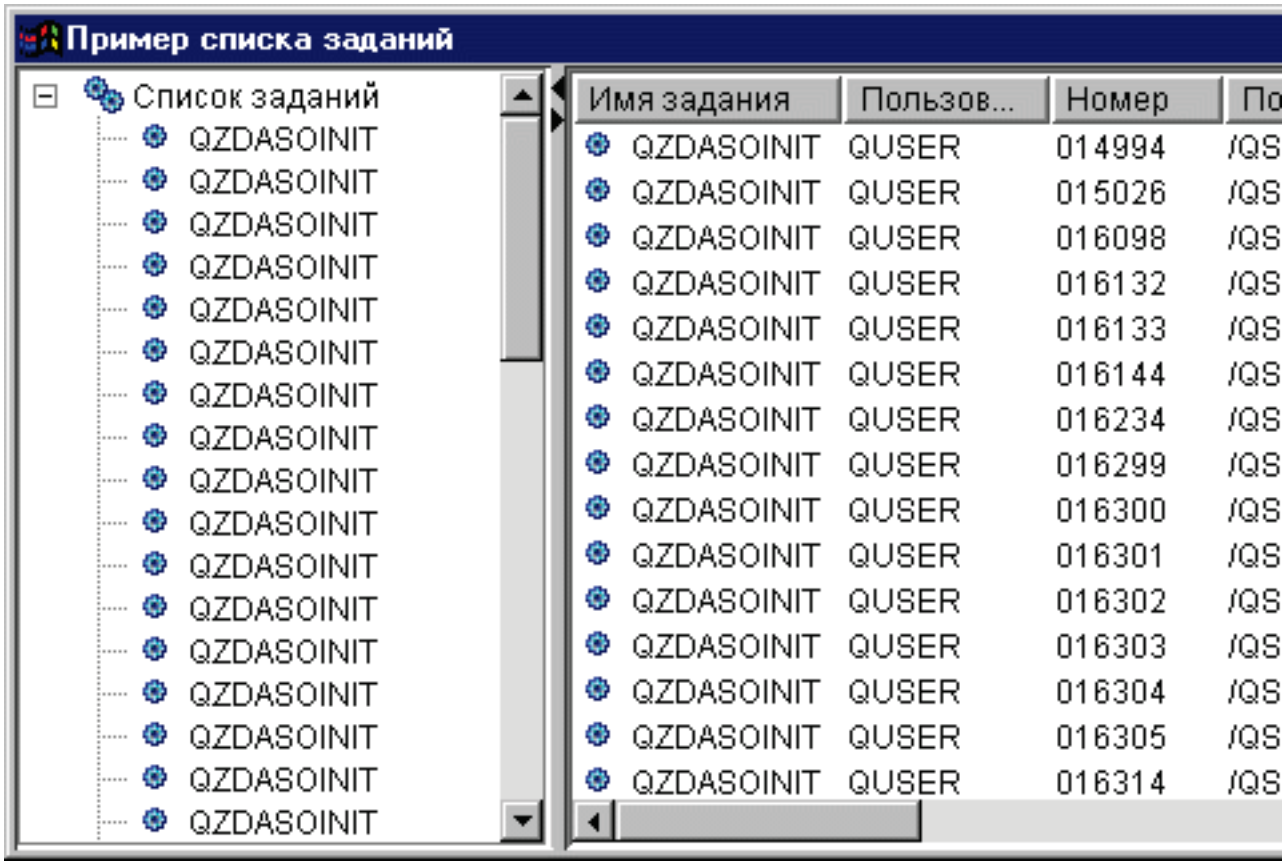
Ниже приведен пример создания объекта VJobList и его представления в окне объекта AS400ExplorerPane:

```
                    // Создание объекта VJobList.
// Предполагается, что объект AS400
                    // уже создан и инициализирован.
     VJobList root = new VJobList (system);

                    // Создание и загрузка объекта
                    // AS400ExplorerPane.
     AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
     explorerPane.load ();

                    // Добавление панели проводника во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
     frame.getContentPane ().add (explorerPane);
```

Примеры

Данный пример VJobList создает панель AS400ExplorerPane со списком заданий. В список включены задания с указанным именем.

На следующем рисунке показан компонент GUI VJobList:



VJobList Javadoc

“Класс AS400Panes” на стр. 253
Объекты AS400Pane - это компоненты пакета Vaccess, предназначенные для работы с ресурсами сервера с помощью GUI. Для каждого типа ресурсов предусмотрен свой набор допустимых операций.

VJob Javadoc

Классы сообщений Vaccess

С помощью компонентов GUI, предназначенных для работы с сообщениями, программы на Java могут выводить списки сообщений сервера в окнах GUI.

Предусмотрены следующие компоненты:

- Объект VMessageList - это ресурс, представляющий список сообщений. Он рассчитан на применение с объектами AS400Panes и предназначен для списков сообщений, выданных в ходе выполнения команд и программ системы AS/400.
- Объект VMessageQueue - это ресурс, представляющий сообщения из очередей сообщений сервера. Он также рассчитан на применение с объектами AS400Panes.

AS400Panes - это компоненты GUI, предназначенные для работы с ресурсами сервера. Объекты VMessageList и VMessageQueue рассчитаны на применение с этими объектами.

Объекты AS400Panes, VMessageList и VMessageQueue - это универсальный инструментарий для выполнения всевозможных операций над сообщениями.

Класс VMessageList:

A VMessageList object is a resource that represents a list of messages for use in AS400Panes. Он предназначен для списков сообщений, выданных в ходе выполнения команд и программ системы AS/400.

Списки сообщений можно получать с помощью следующих методов:

- CommandCall.getMessageList()
- CommandCallButton.getMessageList()
- CommandCallMenuItem.getMessageList()
- ProgramCall.getMessageList()
- ProgramCallButton.getMessageList()
- ProgramCallMenuItem.getMessageList()

Для работы с объектом VMessageList нужно задать свойство messageList. Set this property by using a constructor or through the setMessageList() method. Затем нужно сделать VMessageList корневым объектом объекта AS400Pane с помощью конструктора или метода setRoot() объекта AS400Pane.

Сразу после создания объекты AS400Pane и VMessageList находятся в стандартном начальном состоянии. Список сообщений не загружается во время создания объекта. Для загрузки данных нужно явно вызвать метод load() для одного из объектов.

Во время работы программы пользователь может выполнять действия над сообщениями с помощью меню, появляющихся при нажатии правой кнопки мыши. В этих меню может быть предусмотрен пункт **Свойства**, предназначенный для просмотра свойств (серьезности, типа, даты и т.п.).

С помощью метода setAllowActions() вы можете ограничить число действий, которые смогут выполнять пользователи.

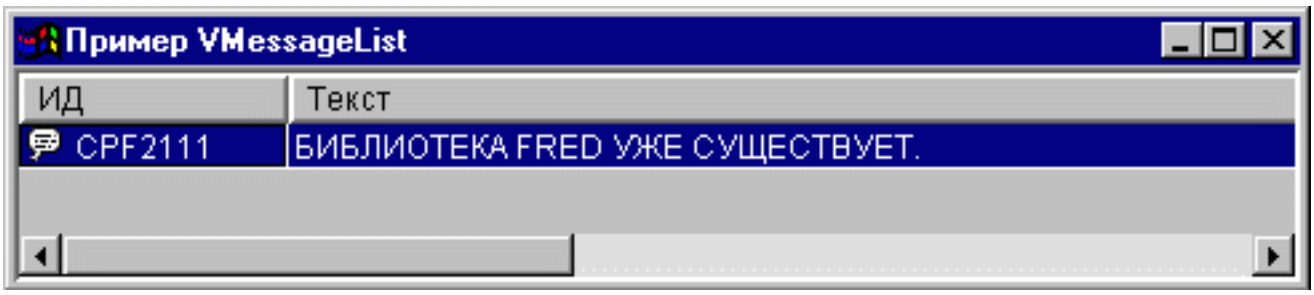
Следующий фрагмент кода создает объект VMessageList для сообщений, выданных в ходе выполнения команды, и выводит их в панели AS400DetailsPane:

```
                    // Создание объекта VMessageList.
                    // Предполагается, что объект
                    // CommandCall (command) уже создан и
                    // инициализирован.
    VMessageList root = new VMessageList (command.getMessageList ());

                    // Создание и загрузка объекта
                    // AS400DetailsPane.
    AS400DetailsPane detailsPane = new AS400DetailsPane (root);
    detailsPane.load ();

                    // Добавление панели со сведениями во фрейм.
                                    // Предполагается, что фрейм JFrame
                    // уже создан.
    frame.getContentPane ().add (detailsPane);
```

Пример

Показывает список сообщений команды на панели AS400DetailsPane с помощью объекта VMessageList. На рис. 1 показан компонент GUI VMessageList:

Рисунок 1: Компонент GUI VMessageList



Ссылки, связанные с данной

“Класс AS400Panes” на стр. 253
Объекты AS400Pane - это компоненты пакета Vaccess, предназначенные для работы с ресурсами сервера с помощью GUI. Для каждого типа ресурсов предусмотрен свой набор допустимых операций.

Информация, связанная с данной

VMessageList Javadoc

Класс VMessageQueue:

A VMessageQueue object is a resource that represents the messages in a server message queue for use in AS400Panes.

Для работы с объектом VMessageQueue необходимо задать свойства system и path. с помощью конструктора класса или методов setSystem() и setPath(). Затем нужно сделать VMessageQueue корневым объектом объекта AS400Pane с помощью конструктора или метода setRoot() объекта AS400Pane.

В объекте VMessageQueue предусмотрены свойства, позволяющие определять наборы сообщений для представления в объектах AS400Pane. Use setSeverity() to specify the severity of messages that appear. Use setSelection() to specify the type of messages that appear.

Сразу после создания объекты AS400Pane и VMessageQueue находятся в стандартном начальном состоянии. Список сообщений не загружается при создании объекта. Для загрузки его содержимого необходимо явно вызвать метод load() для одного из объектов. После этого все данные списка будут загружены с сервера.

Во время работы программы пользователь может выполнять действия над сообщениями и очередями сообщений с помощью меню, появляющихся при нажатии правой кнопки мыши. В контекстном меню очереди сообщений могут быть предусмотрены следующие пункты:

- **Очистить** - очистка очереди сообщений
- **Свойства** - просмотр и изменение серьезности сообщений и параметров отбора. С помощью этой опции можно изменить содержимое списка.

Предусмотрены следующие действия над сообщениями в очереди сообщений:

- **Удалить** - удаление сообщения из очереди
- **Ответ** - отправка ответа на сообщение-вопрос
- **Свойства** - просмотр свойств (уровня серьезности, типа и даты)

Пользователи могут работать только с теми сообщениями, к которым у них есть права доступа. Кроме того, с помощью метода setAllowActions() можно запретить пользователям выполнять действия над объектами.

Следующий фрагмент кода создает объект VMessageQueue и выводит его содержимое в панели AS400ExplorerPane:

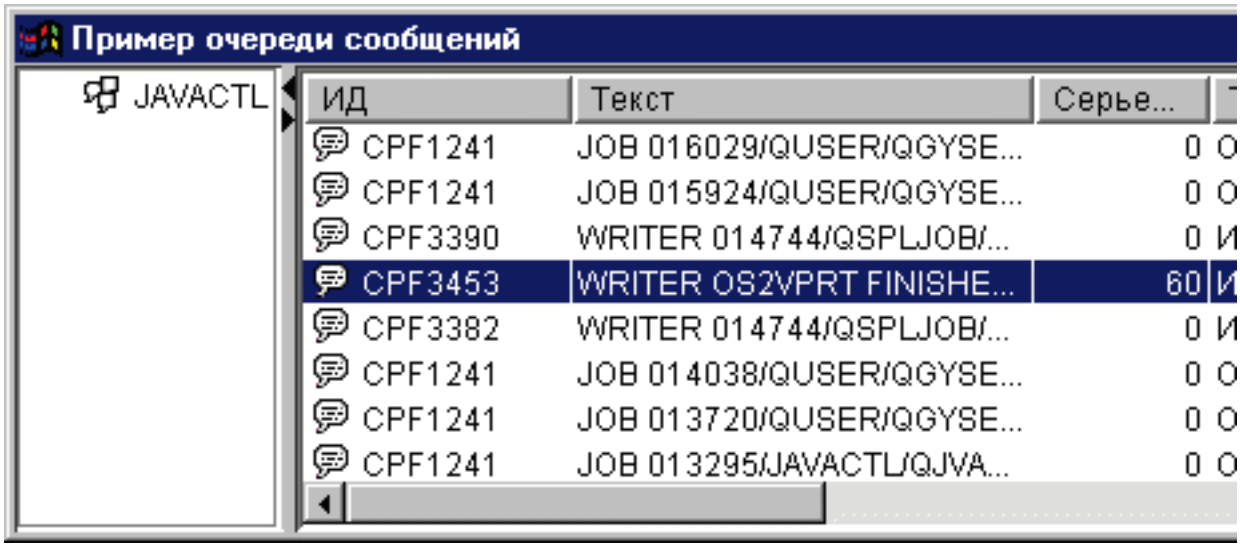
```
// Создание объекта VMessageQueue.
// Предполагается, что объект AS400
// уже создан и инициализирован.
//
```

```
VMessageQueue root = new VMessageQueue (system, "/QSYS.LIB/MYLIB.LIB/MYMSGQ.MSGQ");

                  // Создание и загрузка объекта
                  // AS400ExplorerPane.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
explorerPane.load ();

                  // Добавление панели проводника во фрейм.
                  // Предполагается, что фрейм JFrame
                  // уже создан.
frame.getContentPane ().add (explorerPane);
```

Пример

Показывает список сообщений очереди в объекте AS400ExplorerPane с помощью объекта VMessageQueue. На рис. 1 показан компонент GUI VMessageQueue:

Рисунок 1: Компонент GUI VMessageQueue



Информация, связанная с данной

VMessageQueue Javadoc

Классы прав доступа

The Permission classes information can be used in a graphical user interface (GUI) through the VIFSFile and VIFSDirectory classes. В обоих классах предусмотрено действие Permission.

Следующий пример иллюстрирует применение действия Permission с классом VIFSDirectory:

```
// Создание объекта AS400
AS400 as400 = new AS400();

// Создание объекта IFSDirectory с заданными
// именем системы и полным путем к объекту QSYS
VIFSDirectory directory = new VIFSDirectory(as400,
                              "/QSYS.LID/testlib1.lib");

// Создание панели проводника
AS400ExplorerPane pane = new AS400ExplorerPane((VNode)directory);

// Загрузка информации
pane.load();
```

Классы печати Vaccess

The vaccess package components listed here allow a Java program to present lists of server print resources in a graphical user interface.

- Объект VPrinters - это ресурс, представляющий список принтеров. Он рассчитан на применение с объектом AS400Pane.
- Объект VPrinter - ресурс, представляющий конкретный принтер и его буферные файлы.
- Объект VPrinterOutput - ресурс, представляющий список буферных файлов.
- Объект SpooledFileViewer - ресурс, предназначенный для визуализации буферных файлов.

Объекты AS400Pane - это компоненты GUI, предназначенные для работы с ресурсами сервера. Объекты VPrinters, VPrinter и VPrinterOutput рассчитаны на применение в панелях AS400Panes.

Объекты AS400Pane, VPrinter, VPrinter и VPrinterOutput - это универсальный инструментарий, позволяющий выполнять широкий спектр операций над ресурсами печати сервера.

Класс VPrinters:

A VPrinters object is a resource that represents a list of printers for use in AS400Panes.

Для работы с объектом VPrinters необходимо правильно задать свойство system. Set this property by using a constructor or through the setSystem() method. Затем нужно сделать VPrinters корневым объектом объекта AS400Pane с помощью конструктора панели или метода setRoot().

В объекте VPrinters предусмотрен метод для определения набора принтеров, которые должны быть представлены в панели AS400Pane. Use setPrinterFilter() to specify a filter that defines which printers should appear.

Сразу после создания объекты AS400Pane и VPrinters находятся в стандартном начальном состоянии. Список принтеров не загружается из системы автоматически. Для загрузки содержимого нужно явно вызвать метод load() для одного из объектов.

Во время работы программы пользователь может выполнять действия над списками принтеров и отдельными принтерами с помощью меню, появляющихся при нажатии правой кнопки мыши. В контекстном меню списка принтеров может быть предусмотрен пункт **Свойства**, позволяющий задавать параметры отбора объектов для списка.

В контекстном меню принтера могут быть предусмотрены следующие пункты:

- **Блокировать** - блокирует принтер
- **Разблокировать** - разблокирует принтер
- **Запустить** - запускает принтер
- **Остановить** - останавливает работу принтера
- **Сделать доступным** - делает принтер доступным
- **Сделать недоступным** - делает принтер недоступным
- **Свойства** - показывает свойства принтера и позволяет задать фильтры

Пользователи могут работать только с теми принтерами, к которым у них есть права доступа. Кроме того, с помощью метода setAllowActions() можно запретить пользователям выполнять действия над объектами.

Следующий фрагмент кода создает объект VPrinters и выводит его в панели AS400TreePane:

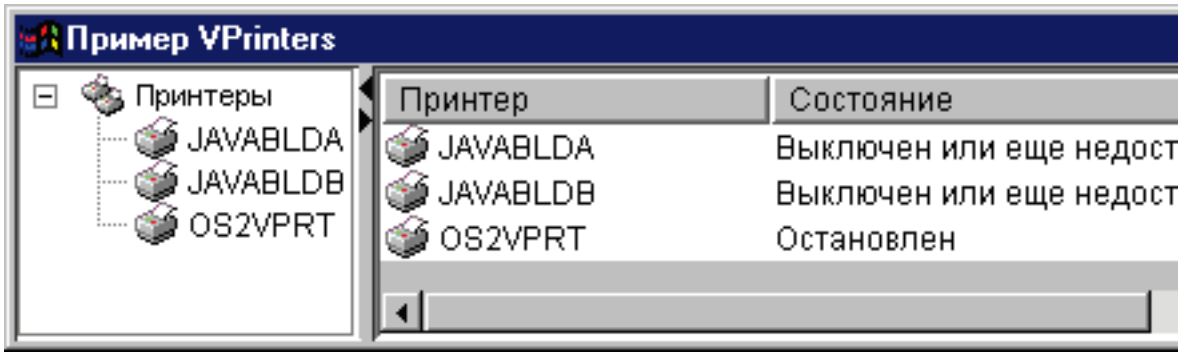
```
                    // Создание объекта VPrinters.
                    // Предполагается, что объект AS400
                    // уже создан и инициализирован.
                    //
    VPrinters root = new VPrinters (system);
```

```
                    // Создание и загрузка объекта
                    // AS400TreePane.
AS400TreePane treePane = new AS400TreePane (root);
treePane.load ();

                    // Добавление панели во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
frame.getContentPane ().add (treePane);
```

Пример

Показывает ресурсы печати на панели AS400ExplorerPane с помощью объекта VPrinters. На рис. 1 показан компонент GUI VPrinters:

Рисунок 1: Компонент GUI VPrinters



Информация, связанная с данной

VPrinters Javadoc

Класс VPrinter:

A VPrinter object is a resource that represents a server printer and its spooled files for use in AS400Panes.

Для работы с объектом VPrinter необходимо правильно задать свойство printer. Set this property by using a constructor or through the setPrinter() method. Затем нужно сделать VPrinter корневым объектом объекта AS400Pane с помощью конструктора панели или метода setRoot().

Сразу после создания объекты AS400Pane и VPrinter находятся в стандартном начальном состоянии. При создании объекта VPrinter в него не загружаются информация о принтере и список буферных файлов.

Для загрузки содержимого необходимо явно вызвать метод load() для одного из объектов. После этого все данные будут загружены с сервера.

Во время работы программы пользователь может выполнять действия над принтерами и объектами вывода с помощью меню, появляющихся при нажатии правой кнопки мыши. В контекстном меню очереди сообщений могут быть предусмотрены следующие пункты:

- **Блокировать** - блокирует принтер
- **Разблокировать** - разблокирует принтер
- **Запустить** - запускает принтер
- **Остановить** - останавливает работу принтера
- **Сделать доступным** - делает принтер доступным
- **Сделать недоступным** - делает принтер недоступным
- **Свойства** - показывает свойства принтера и позволяет задать фильтры

В контекстном меню буферных файлов могут быть предусмотрены следующие пункты:

- **Ответить** - отправка ответа в буферный файл
- **Блокировать** - блокирование буферного файла
- **Разблокировать** - разблокирование буферного файла
- **Печатать следующий** - печатает следующий буферный файл
- **Отправить** - отправка буферного файла
- **Переместить** - перемещение буферного файла
- **Удалить** - удаление буферного файла
- **Свойства** - показывает свойства буферного файла и позволяет изменить некоторые из них

Пользователи могут работать только с теми принтерами и буферными файлами, к которым у них есть права доступа. Кроме того, с помощью метода setAllowActions() можно запретить пользователям выполнять действия над объектами.

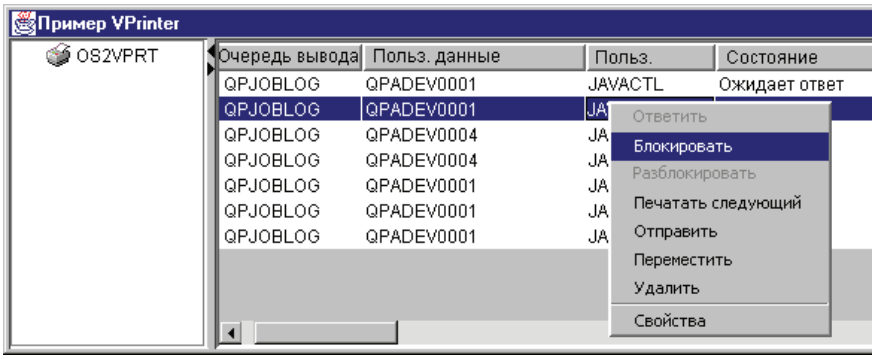
Следующий фрагмент кода создает объект VPrinter и показывает его в панели AS400ExplorerPane:

```
                    // Создание объекта VPrinter.
                    // Предполагается, что объект AS400
                    // уже создан и инициализирован.
                    //
VPrinter root = new VPrinter (new Printer (system, "MYPRINTER"));

                    // Создание и загрузка объекта
                    // AS400ExplorerPane.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (root);
explorerPane.load ();

                    // Добавление панели проводника во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
frame.getContentPane ().add (explorerPane);
```

Example

Показывает список ресурсов печати на панели AS400ExplorerPane с помощью объекта VPrinter. На рис. 1 показан компонент GUI VPrinter:

Рисунок 1: Компонент GUI VPrinter



Информация, связанная с данной

VPrinter Javadoc

Класс VPrinterOutput:

A VPrinterOutput object is a resource that represents a list of spooled files on a server for use in AS400Panes.

Для работы с объектом VPrinterOutput необходимо задать свойство system. This property can be set using a constructor or through the setSystem() method. Затем нужно сделать VPrinterOutput корневым объектом объекта AS400Pane с помощью конструктора или метода setRoot() объекта AS400Pane.

В объекте VPrinterOutput предусмотрены свойства, позволяющие определить набор буферных файлов для представления в объектах AS400Pane. Use setFormTypeFilter() to specify which types of forms should appear. Use setUserDataFilter() to specify which user data should appear. Finally, use setUserFilter() to specify which users spooled files should appear.

Сразу после создания объекты AS400Pane и VPrinterOutput находятся в стандартном начальном состоянии. При создании объекта VPrinter в него не загружается список буферных файлов. Для загрузки содержимого необходимо явно вызвать метод load() для одного из объектов. После этого все данные будут загружены с сервера.

Во время работы программы пользователь может выполнять действия над объектами вывода и списками объектов вывода с помощью меню, появляющихся при нажатии правой кнопки мыши. В контекстном меню может быть предусмотрен пункт **Свойства**, позволяющий задавать параметры отбора объектов для списка.

В контекстном меню буферного файла могут быть предусмотрены следующие пункты:

- **Ответить** - отправка ответа в буферный файл
- **Блокировать** - блокирование буферного файла
- **Разблокировать** - разблокирование буферного файла
- **Печатать следующий** - печатает следующий буферный файл
- **Отправить** - отправка буферного файла
- **Переместить** - перемещение буферного файла
- **Удалить** - удаление буферного файла
- **Свойства** - показывает свойства буферного файла и позволяет изменить некоторые из них

Пользователи могут работать только с теми буферными файлами, к которым у них есть права доступа. Кроме того, с помощью метода setAllowActions() можно запретить пользователям выполнять действия над объектами.

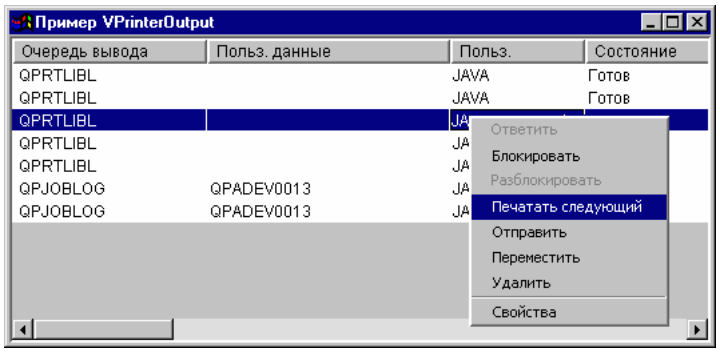
Следующий фрагмент кода создает объект VPrinterOutput и выводит его в панели AS400ListPane:

```
                    // Создание объекта VPrinterOutput.
                    // Предполагается, что объект AS400
                    // уже создан и инициализирован.
                    //
VPrinterOutput root = new VPrinterOutput (system);

                    // Создание и загрузка объекта
                    // AS400ListPane.
AS400ListPane listPane = new AS400ListPane (root);
listPane.load ();

                    // Добавление панели во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
frame.getContentPane ().add (listPane);
```

Example

Показывает список буферных файлов с помощью объекта VPrinterOutput. На рис. 1 показан компонент GUI VPrinterOutput:

Рисунок 1: Компонент GUI VPrinterOutput



Информация, связанная с данной

VPrinterOutput Javadoc

Класс SpooledFileViewer:

The IBM Toolbox for Java SpooledFileViewer class creates a window for viewing Advanced Function Printing (AFP) and Systems Network Architecture character string (SCS) files that have been spooled for printing.

Таким образом, этот класс предоставляет функцию просмотра буферных файлов, широко распространенную в программах текстовой обработки. Пример применения класса показан на рис. 1.

Программа просмотра буферных файлов чаще всего применяется в тех случаях, когда требуется просмотреть формат документа или данные, не печатая их, или когда принтер недоступен.

Примечание: На сервере хоста должен быть установлен компонент 8 SS1 (Совместимые шрифты AFP).

Применение класса SpooledFileViewer

Для создания экземпляра класса SpooledFileViewer можно воспользоваться одним из трех конструкторов. The SpooledFileViewer() constructor can be used to create a viewer without a spooled file associated with it. If this constructor is used, a spooled file will need to be set later using setSpooledFile(SpooledFile). The SpooledFileViewer(SpooledFile) constructor can be used to create a viewer for the given spooled file, with page one as the initial view. Finally, the SpooledFileViewer(spooledFile, int) constructor can be used to create a viewer for the given spooled file with the specified page as the initial view. No matter which constructor is used, once a viewer is created, a call to load() must be performed in order to actually retrieve the spooled file data.

Для просмотра отдельных страниц буферного файла предусмотрены следующие методы:

- load FlashPage()
- load Page()
- pageBack()
- pageForward()

Если вам требуется более тщательно изучить некоторые разделы документа, измените размер страницы с помощью одного из следующих методов:

- fitHeight()
- fitPage()
- fitWidth()
- actualSize()

Your program concludes with calling the close() method that closes the input stream and releases any resource associations with the stream.

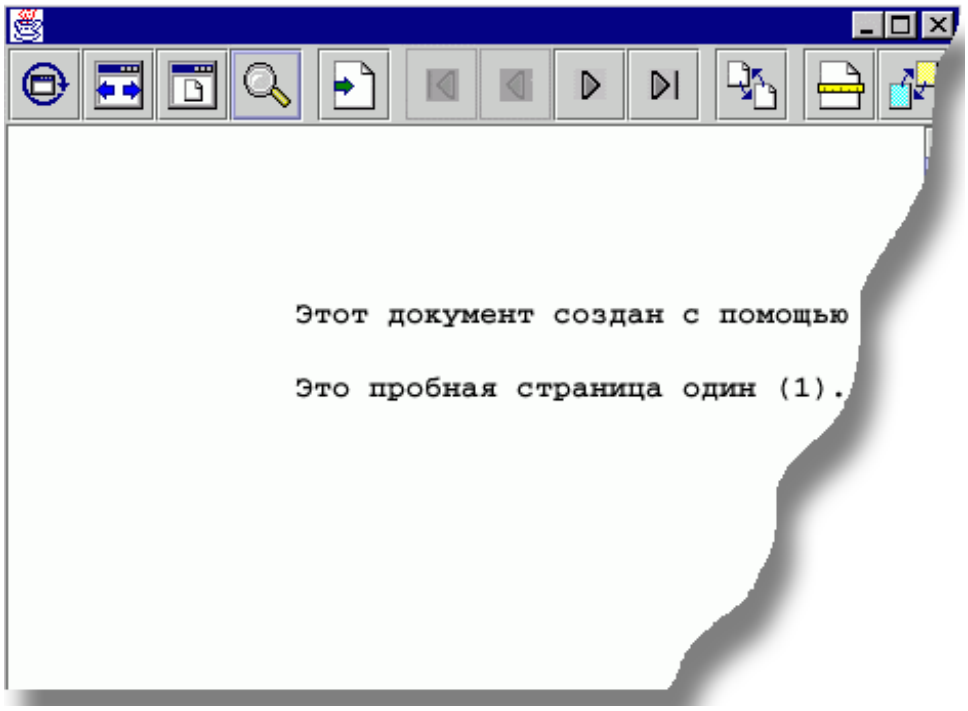
Применение класса SpooledFileViewer

Экземпляр класса SpooledFileViewer является графическим средством просмотра, которое позволяет работать с буферными файлами AFP и SCS. Например, ниже приведен фрагмент кода, в котором создается программа просмотра, показанная на рисунке 1. Эта программа выводит буферный файл, созданный на сервере ранее.

Примечание: Для просмотра описания панели нажмите соответствующую кнопку на рисунке 1 , либо (если ваш браузер не поддерживает JavaScript), просмотрите описание панели инструментов.

```
// Пусть splf - имя буферного файла.
// Создание программы просмотра буферного файла
SpooledFileViewer splfv = new SpooledFileViewer(splf, 1);
splfv.load();
// Вывод окна программы просмотра в главном окне
JFrame frame = new JFrame("My Window");
frame.getContentPane().add(splfv);
frame.pack();
frame.show();
```

Рисунок 1: Класс SpooledFileViewer



Описание панели инструментов SpooledFileViewer


Эта кнопка позволяет просмотреть исходный размер изображения страницы буферного файла с помощью метода actualSize().


Эта кнопка позволяет растянуть изображение страницы буферного файла на всю ширину фрейма окна просмотра с помощью метода fitWidth().

 Эта кнопка позволяет растянуть изображение страницы буферного файла на всю высоту фрейма окна просмотра с помощью метода fitPage().

 Эта кнопка позволяет увеличить или уменьшить размер изображения страницы буферного файла с помощью установки одного из готовых значений масштаба или любого необходимого значения в текстовом поле окна диалога, которое появляется после выбора этого инструмента.

 Эта кнопка позволяет перейти на любую страницу буферного файла.

 Эта кнопка позволяет перейти к первой странице буферного файла; если она неактивна, это означает, что на экране показана первая страница.

 Эта кнопка позволяет перейти от текущей страницы к предыдущей.

 С помощью этой кнопки можно перейти от текущей страницы к следующей.

 Эта кнопка позволяет перейти к последней странице буферного файла; если она неактивна, это означает, что на экране показана последняя страница.

 Эта кнопка позволяет загрузить страницу, которую пользователь просматривал перед текущей, с помощью метода loadFlashPage().

 С помощью этой кнопки можно задать размер применяемой бумаги.

 С помощью этой кнопки можно задать точность просмотра страниц буферного файла.

SpooledFileViewer Javadoc

Классы ProgramCall Vaccess

С помощью компонентов Vaccess, предназначенных для вызова программ, программы на Java могут создавать кнопки и меню для вызова программ сервера. Input, output, and input/output parameters can be specified using ProgramParameter objects. В выходных параметрах и параметрах смешанного типа возвращается результат выполнения программы сервера.

A ProgramCallButton object represents a button that calls an server program when pressed. Класс ProgramCallButton расширяет класс JButton из комплекта Java Foundation Classes (JFC), поэтому данная кнопка выглядит стандартно.

Similarly, a ProgramCallMenuItem object represents a menu item that calls an server program when selected. Класс ProgramCallMenuItem расширяет класс JMenuItem из комплекта JFC, поэтому все пункты меню выглядят стандартно.

Для работы с компонентами Vaccess, предназначенными для вызова программ, нужно задать свойства system и program с помощью конструктора класса или методов setSystem() и setProgram().

В приведенном ниже примере создается объект ProgramCallMenuItem. Если при выполнении программы пользователь выберет этот пункт меню, будет запущена программа.

```
                    // Создание объекта ProgramCallMenuItem.
                    // Предполагается, что объект AS400
// (system) уже создан и
                    // инициализирован. Пункт меню будет
                    // называться "Выбери меня", и значка у него
                    // не будет.
     ProgramCallMenuItem menuItem = new ProgramCallMenuItem ("Выбери меня", null, system);

                    // Создание объекта полного имени, указывающего
                    // на программу MYPROG из
                    // библиотеки MYLIB.
     QSYSObjectPathName programName = new QSYSObjectPathName("MYLIB", "MYPROG", "PGM");

                    // Задание имени программы.
     menuItem.setProgram (programName.getPath());

                    // Добавление пункта в меню.
                    // Предполагается, что меню
                    // уже создано.
     menu.add (menuItem);
```

Во время выполнения программы сервера могут выдавать сообщения. To detect when the server program runs, add an ActionCompletedListener to the button or menu item using the addActionCompletedListener() method. When the program runs, it fires an ActionCompletedEvent to all such listeners. Обработчик событий может получать сообщения, выдаваемые программой сервера, с помощью метода getMessageList().

В этом примере создается объект ActionCompletedListener, который обрабатывает все сообщения сервера, созданные программой:

```
                    // Добавление обработчика ActionCompletedListener,
                    // реализованного с помощью анонимного
                    // внутреннего класса. Это наиболее удобный способ
// создания несложных обработчиков
                    // событий.
     menuItem.addActionCompletedListener (new ActionCompletedListener ()
     {
          public void actionCompleted (ActionCompletedEvent event)
          {
                    // Отправка исходного кода события в
                    // ProgramCallMenuItem.
               ProgramCallMenuItem sourceMenuItem = (ProgramCallMenuItem) event.getSource ();

                    // Получение списка сообщений, выданных
                    // программой.
               AS400Message[] messageList = sourceMenuItem.getMessageList ();

                    // ... Обработка списка сообщений.
          }
     });
```

Параметры

ProgramParameter objects are used to pass parameter data between the Java program and the server program. Input data is set with the setInputData() method. After the program is run, output data is retrieved with the getOutputData() method.

Каждый параметр представляет собой массив байтов. Приведение данных к нужным форматам Java и сервера должна выполнить программа на Java. Классы преобразования данных содержат методы преобразования данных.

В компонент интерфейса GUI параметры можно добавлять либо по одному (метод addParameter()), либо все сразу (метод setParameterList()).

Дополнительная информация об объектах ProgramParameter приведена в разделе класс доступа ProgramCall.

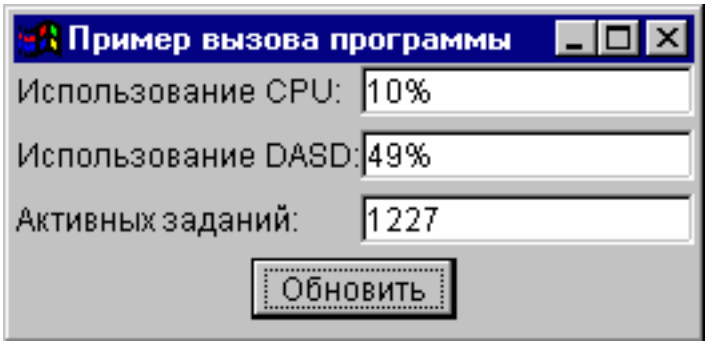
В приведенном ниже примере добавляются два параметра:

```
                    // Первый параметр - строка длиной
                    // до 100 символов.
                    // Это входной параметр. Предполагается,
                    // Предполагается, что "name" - это объект типа String,
                    // который уже создан и инициализирован.
AS400Text parm1Converter = new AS400Text (100, system.getCcsid (), system);
ProgramParameter parm1 = new ProgramParameter (parm1Converter.toBytes (name));
menuItem.addParameter (parm1);

                    // Второй параметр - целое число.
                    // Это выходной параметр.
AS400Bin4 parm2Converter = new AS400Bin4 ();
ProgramParameter parm2 = new ProgramParameter (parm2Converter.getByteLength ());
menuItem.addParameter (parm2);

                    // ... Определение значения
                    // выходного параметра после
                    // выполнения программы.
int result = parm2Converter.toInt (parm2.getOutputData ());
```

Примеры

Пример применения объекта ProgramCallButton в приложении. На рис. 1 показан внешний вид объекта ProgramCallButton:

Рисунок 1: Применение объекта ProgramCallButton в приложении



ProgramParameter Javadoc

ProgramCallButton Javadoc

ProgramCallMenuItem Javadoc

ActionCompletedListener Javadoc

ActionCompletedEvent Javadoc

Классы Vaccess для доступа на уровне записей

С помощью классов для доступа на уровне записей, входящих в пакет Vaccess, программы на Java могут работать с файлами сервера.

Предусмотрены следующие компоненты:

- Объект RecordListFormPane - этот ресурс показывает записи файла сервера в виде формы.
- Объект RecordListTablePane - этот ресурс показывает записи файла сервера в виде таблицы.
- Объект RecordListTableModel - этот ресурс предназначен для работы с записями файла сервера с помощью таблицы.

Доступ по ключу

Компоненты доступа на уровне записей поддерживают режим доступа к файлам сервера по ключу. Доступом по ключу называется режим доступа программы на Java, при котором записи в файлах упорядочены по специальным значениям - ключам.

Для пользователя работа в режиме доступа по ключу не отличается от работы в режиме обычного доступа на уровне записей. Для включения режима доступа по ключу нужно вызвать метод setKeyed(). Ключ можно задать с помощью конструктора или метода setKey(). Дополнительная информация о способах задания ключа приведена в соответствующем разделе.

По умолчанию выдаются только записи, у которых ключ совпадает с указанным значением. Однако существуют и другие режимы, которые можно включать с помощью свойства searchType метода setSearchType(). Они перечислены ниже:

- KEY_EQ - Выдаются записи с ключом, равным указанному значению.
- KEY_GE - Выдаются записи с ключом, большим или равным указанному значению.
- KEY_GT - Выдаются записи с ключом, большим указанного значения.
- KEY_LE - Выдаются записи с ключом, меньшим или равным указанному значению.
- KEY_LT - Выдаются записи с ключом, меньшим указанного значения.

Следующий фрагмент кода создает объект RecordListTablePane и показывает записи с ключом, меньшим или равным указанному значению.

```
                    // Создание ключа, равного
                    // целому числу 5.
Object[] key = new Object[1];
key[0] = new Integer (5);

                    // Создание объекта RecordListTablePane.
                    // Предполагается, что объект "system"
                    // уже создан и инициализирован.
                    // Указание ключа и типа
                    // поиска записей.
RecordListTablePane tablePane = new RecordListTablePane (system,
     "/QSYS.LIB/QGPL.LIB/PARTS.FILE", key, RecordListTablePane.KEY_LE);

                    // Загрузка содержимого файла.
tablePane.load ();

                    // Добавление панели с таблицей во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
frame.getContentPane ().add (tablePane);
```

Класс RecordListFormPane:

A RecordListFormPane presents the contents of a server file in a form. В каждый момент времени в форме показана одна запись; пользователь может перемещаться по списку и обновлять его содержимое с помощью специальных кнопок.

Для работы с объектом RecordListFormPane необходимо задать свойства system и fileName. Set these properties by using the constructor or the setSystem() and setFileName() methods. Use load() to retrieve the file contents and present the first record. When the file contents are no longer needed, call close() to ensure that the file is closed.

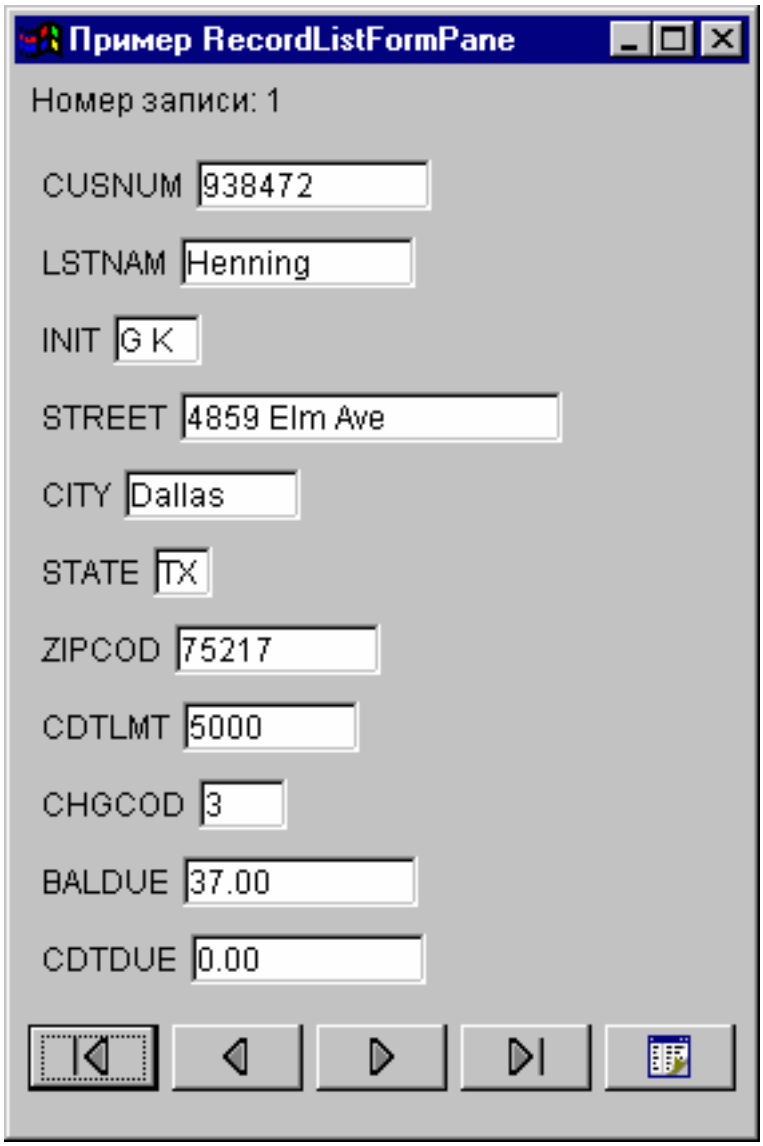
Следующий фрагмент кода создает объект RecordListFormPane и добавляет его во фрейм:

```
                    // Создание объекта RecordListFormPane.
                    // Предполагается, что объект AS400
// уже создан и инициализирован.
                    //
     RecordListFormPane formPane = new RecordListFormPane (system,
                                   "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                    // Загрузка содержимого файла.
     formPane.load ();

                    // Добавление панели с формой во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
     frame.getContentPane ().add (formPane);
```

Пример

Просмотр содержимого файла с помощью объекта RecordListFormPane. На рис. 1 показан компонент GUI RecordListFormPane:

Рисунок 1: Компонент GUI RecordListFormPane



Информация, связанная с данной

RecordListFormPane Javadoc

Класс RecordListTablePane:

A RecordListTablePane presents the contents of a server file in a table. Каждая строка таблицы соответствует одной записи файла, а каждый столбец строки - одному полю.

Для работы с объектом RecordListTablePane необходимо задать свойства system и fileName. Set these properties by using the constructor or the setSystem() and setFileName() methods. Use load() to retrieve the file contents and present the records in the table. When the file contents are no longer needed, call close() to ensure that the file is closed.

Следующий фрагмент кода создает объект RecordListTablePane и добавляет его во фрейм:

```
                    // Создание объекта RecordListTablePane.
                    // Предполагается, что объект AS400
// уже создан и инициализирован.
                    //
     RecordListTablePane tablePane = new RecordListTablePane (system,
                                  "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");
                    // Загрузка содержимого файла.
```

```
tablePane.load ();

                    // Добавление панели с таблицей во фрейм.
                    // Предполагается, что фрейм JFrame
                    // уже создан.
frame.getContentPane ().add (tablePane);
```

Информация, связанная с данной

RecordListTablePane Javadoc

Классы RecordListTablePane и RecordListTableModel:

Объект RecordListTablePane реализован на основе принципа ″модель-визуализация-управление″, согласно которому данные и пользовательский интерфейс должны быть разнесены в разные классы.

The implementation integrates RecordListTableModel with Java Foundation Classes' (JFC) JTable. Класс RecordListTableModel отвечает за загрузку содержимого файла, а объект JTable - за графическое представление данных.

Средств объекта RecordListTablePane достаточно для выполнения большинства практических задач. Если же вам требуется более тонкое управление компонентом JFC, воспользуйтесь объектом RecordListTableModel напрямую. Еще одно преимущество заключается в том, что данный объект можно применять с разными компонентами GUI.

Для работы с объектом RecordListTableModel необходимо задать свойства system и fileName. Set these properties by using the constructor or the setSystem() and setFileName() methods. Use load() to retrieve the file contents. When the file contents are no longer needed, call close() to ensure that the file is closed.

Следующий фрагмент кода создает объект RecordListTableModel и показывает его в таблице JTable:

```
                    // Создание объекта RecordListTableModel.
                    // Предполагается, что объект AS400
// уже создан и инициализирован.
                    //
     RecordListTableModel tableModel = new RecordListTableModel (system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

                    // Загрузка содержимого файла.
     tableModel.load ();

                    // Создание объекта JTable для модели.
     JTable table = new JTable (tableModel);

                    // Добавление таблицы во фрейм.
// Предполагается, что фрейм JFrame
                    // уже создан.
     frame.getContentPane ().add (table);
```

RecordListTableModel Javadoc

Классы ResourceListPane и ResourceListDetailsPane

Use the ResourceListPane and ResourceListDetailsPane classes to present a resource list in a graphical user interface (GUI).

- Класс ResourceListPane представляет список ресурсов в виде списка javax.swing.JList. Каждый элемент списка представляет один объект из списка ресурсов.
- Класс ResourceListDetailsPane представляет список ресурсов в виде таблицы javax.swing.JTable. Каждая строка таблицы содержит информацию об одном объекте из списка ресурсов.

Столбцы таблицы ResourceListDetailsPane задаются в виде массива атрибутов объектов. Каждому элементу массива атрибутов соответствует один столбец.

По умолчанию как ResourceListPane, так и ResourceListDetailsPane поддерживают всплывающие меню.

Для уведомления о большинстве ошибок применяется класс com.ibm.as400.vaccess.ErrorEvents, а не исключительные ситуации. Для обнаружения и исправления ошибок добавьте программу обработки событий ErrorEvents.

Пример: Просмотр списка ресурсов в окне графического интерфейса

Ниже приведен пример создания объекта ResourceList с информацией о ресурсах всех пользователей системы и его вывода на панель:

```
R    // Создание списка ресурсов.
R    AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
R    RUserList userList = new RUserList(system);
R
R    // Создание панели ResourceListDetailsPane.
R // В этом примере таблица содержит два столбца.
R // contains the icons and names for each user.     // Второй столбец содержит описание, связанное с
R    // пользователем.
R    Object[] columnAttributeIDs = new Object[] { null, RUser.TEXT_DESCRIPTION };
R    ResourceListDetailsPane detailsPane = new ResourceListDetailsPane();
R    detailsPane.setResourceList(userList);
R    detailsPane.setColumnAttributeIDs(columnAttributeIDs);
R
R    // Добавление объекта ResourceListDetailsPane во фрейм JFrame и вывод фрейма.
R    JFrame frame = new JFrame("My Window");
R    frame.getContentPane().add(detailsPane);
R    frame.pack();
R    frame.show();
R
R    // Необходимо загрузить данные в объект ResourceListDetailsPane.
R    // Данные из системы iSeries можно получить
R // of users is retrieved from the server.
R    detailsPane.load ();
```

ResourceListPane Javadoc

ResourceListDetailsPane Javadoc

Классы состояния системы

The System status components in the vaccess package allow you to create GUIs by using the existing AS400Panes.

Помимо этого, можно создавать GUI с помощью Java Foundation Classes (JFC). The VSystemStatus object represents a system status on the server. Объект VSystemPool представляет системный пул сервера. Объект VSystemStatusPane - это ресурс, представляющий панель с информацией о состоянии системы.

The VSystemStatus class allows you to get information about the status of a server session within a GUI environment:

- The getSystem() method returns the server where the system status information is contained
- The getText() method returns the description text
- The setSystem() method sets the server where the system status information is located

Помимо вышеуказанных методов, предусмотрены методы просмотра и изменения информации в системном пуле с помощью GUI.

You use VSystemStatus with VSystemStatusPane. Панель VSystemPane может использоваться для работы не только с состоянием системы, но и с информацией системного пула.

VSystemStatus Javadoc

VSystemStatusPane Javadoc

Класс VSystemPool:

The VSystemPool class allows you to retrieve and set system pool information from a server using a GUI design. VSystemPool works with various panes in the vaccess package including the VSystemStatusPane.

Ниже перечислены некоторые методы класса VSystemPool:

- The getActions() method returns a list of actions that you can perform
- The getSystem() method returns the server where the system pool information is found
- The setSystemPool() method sets the system pool object

Информация, связанная с данной

VSystemPool Javadoc

VSystemStatusPane

Класс VSystemStatusPane:

The VSystemStatusPane class allows a Java program to display system status and system pool information.

В класс VSystemStatusPane входят следующие методы:

- getVSystemStatus(): Returns the VSystemStatus information in a VSystemStatusPane.
- setAllowModifyAllPools(): Sets the value to determine if system pool information can be modified.

Следующий фрагмент кода иллюстрирует применение класса VSystemStatusPane:

```
// Создание объекта AS400.
AS400 mySystem = new AS400("mySystem.myCompany.com");

// Создание VSystemStatusPane
VSystemStatusPane myPane = new VSystemStatusPane(mySystem);

// Установка разрешения на изменение пулов
myPane.setAllowModifyAllPools(true);

// Загрузка информации
myPane.load();
```

VSystemStatusPane Javadoc

Системные значения

The system value components in the vaccess package allow a Java program to create GUIs by using the existing AS400Panes or by creating your own panes using the Java Foundation Classes(JFC).

The VSystemValueList object represents a system value list on the server.

To use the System Value GUI component, set the system name with a constructor or through the setSystem() method.

Пример В приведенном ниже примере с помощью панели AS400Explorer создается графический интерфейс для работы с системными значениями:

```
// Создание объекта AS400
AS400 mySystem = new AS400("mySystem.myCompany.com");
VSystemValueList mySystemValueList = new VSystemValueList(mySystem);
as400Panel=new AS400ExplorerPane((VNode)mySystemValueList);
// Создание объекта AS400ExplorerPane и загрузка в него данных
as400Panel.load();
```

Информация, связанная с данной

VSystemValueList Javadoc

Классы Vaccess для работы с пользователями и группами

The users and groups components in the vaccess package allow you to present lists of server users and groups through the VUser class.

Предусмотрены следующие компоненты:

- Объекты AS400Pane - это компоненты GUI, предназначенные для работы с ресурсами сервера.

- A VUserList object is a resource that represents a list of server users and groups for use in AS400Panes.
- A VUserAndGroup object is a resource for use in AS400Panes that represents groups of server users. Программы на Java могут применять его для создания списков всех пользователей, всех групп или всех пользователей, не входящих в группы.

Объекты AS400Pane и VUserList поддерживают различные режимы представления информации. В них предусмотрена возможность выбора пользователей и групп.

Для работы с объектом VUserList нужно задать свойство system. Set this property by using a constructor or through the setSystem() method. Затем нужно сделать VUserList корневым объектом объекта AS400Pane с помощью конструктора или метода setRoot() объекта AS400Pane.

В объекте VUserList предусмотрены свойства, позволяющие определять наборы пользователей и групп для представления в объектах AS400Pane.

- Use the setUserInfo() method to specify the types of users that should appear.
- Use the setGroupInfo() method to specify a group name.

You can use the VUserAndGroup object to get information about the Users and Groups on the system. Before you can get information about a particular object, you need to load the information so that it can be accessed. You can display the server in which the information is found by using the getSystem method.

Сразу после создания объекты AS400Pane, VUserList и VUserAndGroup находятся в стандартном начальном состоянии. Список пользователей и групп не загружается из системы автоматически. Для загрузки информации программа на Java должна явно вызвать метод load() для соответствующего объекта.

Щелкнув правой кнопкой мыши на имени пользователя, списке пользователей или имени группы можно открыть контекстное меню. Выберите пункт **Свойства** в меню ярлыков и выполните следующие действия с выбранным объектом:

- Пользователь - Просмотр информации о пользователе, включающей описание, класс пользователя, состояние, описание задания, сведения об объектах вывода, сообщениях, локали, защите и группе пользователя.
- Список пользователей - Работа со свойствами пользователей и групп. Можно также изменить содержимое списка.
- Пользователи и группы - Просмотр свойств, таких как имя и описание пользователя.

Пользователи программы могут работать только с теми пользователями и группами, к которым у них есть права доступа. Более того, с помощью метода setAllowActions() программа на Java может ограничить диапазон действий, разрешенных пользователям.

Следующий фрагмент кода создает объект VUserList и показывает его в панели AS400DetailsPane:

```
// Создание объекта VUserList.
// Предполагается, что объект AS400
// уже создан и инициализирован.
//
VUserList root = new VUserList (system);

// Создание и загрузка объекта
// AS400DetailsPane.
AS400DetailsPane detailsPane = new AS400DetailsPane (root);
detailsPane.load ();

// Добавление панели с подробными сведениями во фрейм.
// Предполагается, что фрейм JFrame
// уже создан.
frame.getContentPane ().add (detailsPane);
```

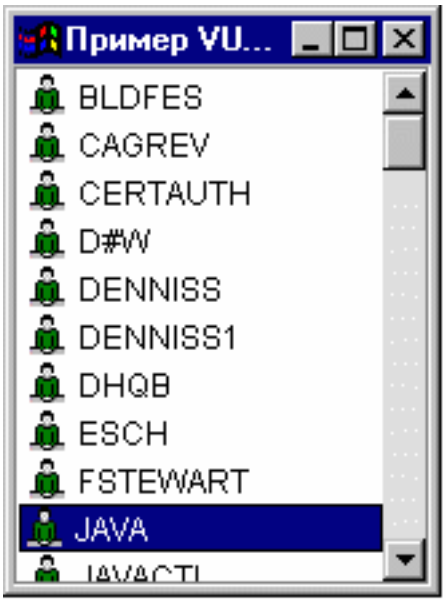
Ниже приведен пример применения объекта VUserAndGroup:

```
// Создать объект VUserAndGroup.
// Предполагается, что объект AS400 уже создан и инициализирован.
VUserAndGroup root = new VUserAndGroup(system);

// Создание и загрузка AS400ExplorerPane.
AS400ExplorerPane explorerPane = new AS400ExplorerPane(root);
explorerPane.load();

// Добавление панели с формой во фрейм.
// Предполагается, что фрейм JFrame уже создан.
frame.getContentPane().add(explorerPane);
```

Другие примеры

Представляет список пользователей в панели AS400ListPane с помощью объекта VUserList.

На следующем рисунке показан компонент GUI VUserList:



VUser Javadoc

VUserAndGroup Javadoc

Graphical Toolbox и PDML

The Graphical Toolbox, a set of UI tools, enables you to create custom user interface panels in Java.

L You can incorporate the panels into your Java applications, applets, or System i Navigator plug-ins. The panels may contain data obtained from the system, or data obtained from another source such as a file in the local file system or a program on the network.

GUI Builder - это визуальный редактор типа WYSIWYG, предназначенный для создания окон диалога, окон свойств и мастеров на языке Java. С помощью этого приложения вы можете добавлять, переносить и изменять управляющие элементы пользовательского интерфейса, расположенные на панели, а также просматривать панель целиком. Созданные определения панелей могут применяться в окнах диалога, окнах свойств и мастерах. Кроме того, их можно добавлять к разделенным панелям, составным панелям и панелям с закладками. Помимо этого, GUI Builder позволяет создавать определения меню, панелей инструментов и контекстных меню. Вы можете добавить объекты JavaHelp в описание панелей, в том числе контекстную справку.

Resource Script Converter преобразует описания ресурсов Windows в формат XML, который применяется в программах на Java . С помощью этой программы вы можете обрабатывать описания ресурсов Windows (файлы .rc) окон диалога и меню Windows. Преобразованные файлы можно затем отредактировать с помощью GUI Builder. Resource Script Converter может применяться в сочетании с GUI Builder для создания окон свойств и мастеров на основе файлов .rc.

Оба рассмотренных средства суть реализация новой технологии, называемой **Язык описаний определений панелей** (**PDML**). Язык PDML основан на Расширяемом языке описаний (XML), не зависит от платформы и предназначен для описания макета элементов пользовательского интерфейса. Определив панели с помощью PDML, вы можете просматривать их с помощью API выполнения, предусмотренного в Graphical Toolbox. При выводе панелей этот API интерпретирует описание PDML и отображает элементы пользовательского интерфейса с помощью классов Java Foundation.

Примечание: Для применения PDML необходима среда выполнения Java (JRE) версии 1.4 или выше.

Преимущества Graphical Toolbox

Сокращает объем кода и ускоряет разработку

Graphical Toolbox значительно ускоряет и упрощает создание пользовательских интерфейсов на языке Java. GUI Builder позволяет контролировать все параметры размещения элементов пользовательского интерфейса в панелях. Поскольку макет описывается на языке PDML, нет необходимости определять интерфейс путем написания кода на языке Java, как и повторно компилировать код в случае изменений. Как следствие, создание и обслуживание приложений на Java занимает значительно меньше времени. С помощью Resource Script Converter вы можете быстро преобразовать большое количество панелей Windows в формат Java.

Создание справки

Определение пользовательских интерфейсов на языке PDML дает и другие преимущества. Вся информация о панели записывается на формальном языке описаний. Это позволяет расширить возможности инструментов и предоставить разработчикам дополнительные функции. Например, и в GUI Builder, и в Resource Script Converter предусмотрена функция создания шаблонов электронной справки по панели в формате HTML. Вам потребуется всего лишь выбрать разделы справки, и они будут автоматически созданы. В шаблон справки автоматически добавляются ссылки на разделы справки. Это означает, что разработчик должен предоставить только саму справочную информацию. Среда выполнения Graphical Toolbox автоматически выдает нужный раздел справки по запросу пользователя.

Автоматическое объединение интерфейса и программного кода

В PDML предусмотрены теги, позволяющие связать управляющий элемент с определенным атрибутом компонента Javabean. После того как вы зададите классы компонентов, содержащие данные для панели, и свяжете их атрибуты с управляющими элементами, соответствующие инструменты смогут автоматически создать шаблон исходного кода на Java для этих компонентов. Во время выполнения Graphical Toolbox управляет обменом данными между указанными компонентами и управляющими элементами панели.

Независимость от платформы

Среда выполнения Graphical Toolbox поддерживает обработку событий, проверку пользовательских данных и стандартные способы обмена данными между управляющими элементами панели. Параметры пользовательского интерфейса на конкретной платформе устанавливаются автоматически в зависимости от текущей операционной системы. С помощью программы GUI Builder вы можете посмотреть, как этот интерфейс будет выглядеть на различных платформах.

Graphical Toolbox содержит два инструмента создания пользовательского интерфейса. Программа GUI Builder позволяет быстро создавать новые панели в визуальной среде, а программа Resource Script Converter - преобразовывать существующие панели Windows в формат Java. Преобразованные файлы можно отредактировать с помощью GUI Builder. Оба инструмента поставляются на национальном языке.

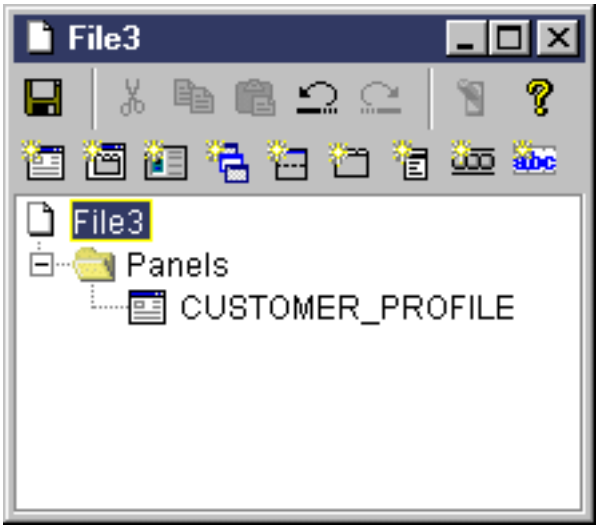
GUI Builder

При запуске программы GUI Builder появляются два окна, показанные на рис. 1:

Рис. 1: Окна GUI Builder

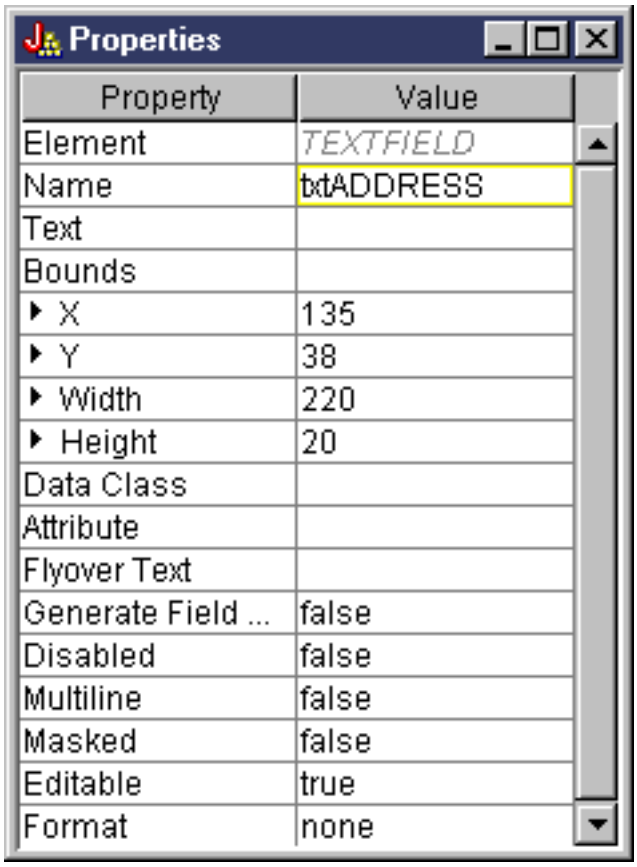
Создание и редактирование файлов PDML с помощью окна Редактор файлов.

Рис. 2: Окно Редактор файлов



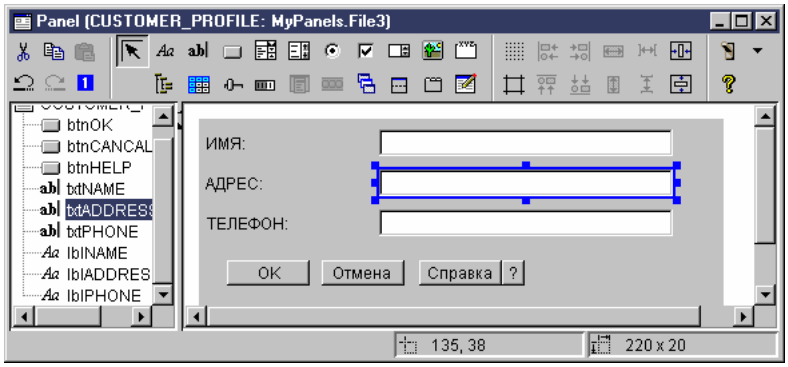
Окно Свойства предназначено для просмотра и изменения свойств выделенного управляющего элемента.

Рис. 3: Окно Свойства



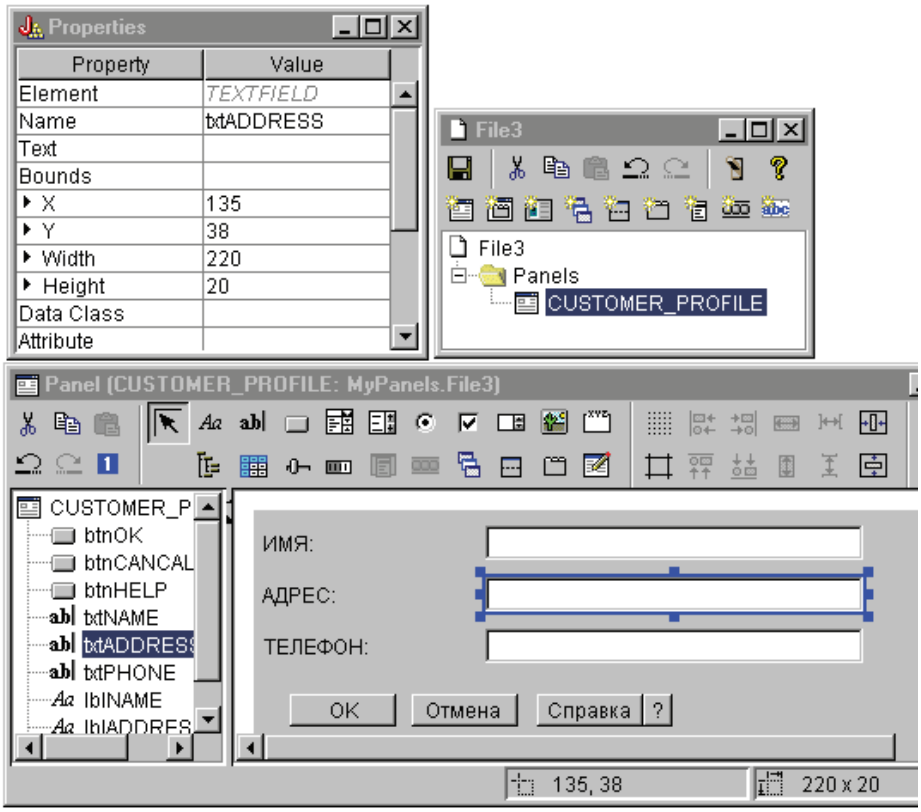
Создание и редактирование компонентов GUI с помощью окна Редактор панелей. Выберите компонент на панели и щелкните мышью в той точке панели, где вы хотите его разместить. На панели инструментов расположены значки для выравнивания группы элементов, предварительного просмотра панели и вызова электронной справки по функциям GUI Builder. Описание назначения каждого значка приведено в разделе Панель инструментов Редактора панелей GUI Builder.

Рис. 4: Окно Редактор панелей



В окне Редактор панелей всегда показана текущая панель, с которой вы работаете. На рис. 5 показана группа окон:

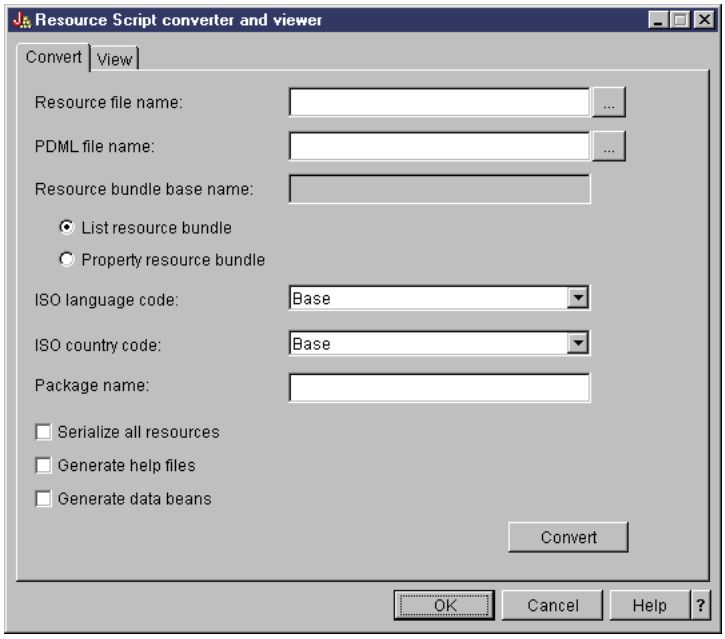
Рис. 5: Группа окон GUI Builder



Resource Script Converter

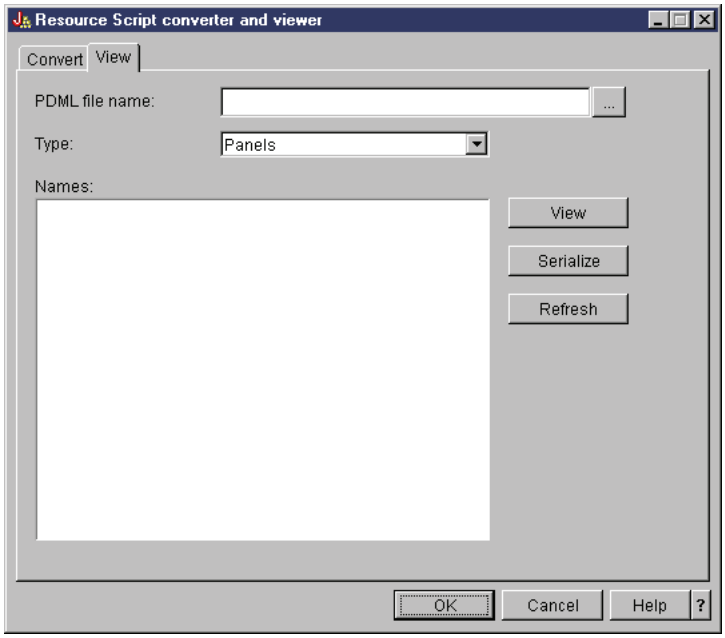
Окно программы Resource Script Converter - это окно диалога с закладками, состоящее из двух панелей. На панели **Преобразовать** вы должны указать имя файла .rc в формате Microsoft или VisualAge for Windows, который необходимо преобразовать в формат PDML. Кроме того, вы можете задать имя целевого файла PDML, а также набор ресурсов Java, который будет содержать преобразованное определение панели. Здесь же вы можете создать шаблон электронной справки по панели, создать шаблон исходного кода Java для объектов с данными для панели и сохранить определение панели в двоичном виде с целью повышения производительности. Подробное описание всех полей панели Преобразовать вы найдете в электронной справке по этой панели.

Рис. 6: Окно программы Resource Script Converter: Панель Преобразовать



После завершения преобразования с помощью панели **Вид** можно просмотреть содержимое созданного файла PDML и новые панели Java. При необходимости вы сможете внести небольшие изменения в панель с помощью программы GUI Builder. Перед преобразованием панели программа Resource Script Converter убеждается в отсутствии файла PDML с описанием панели и сохраняет все изменения для преобразования панели в будущем.

Рис. 7: Окно программы Resource Script Converter: Панель Показать



Настройка Graphical Toolbox

Набор графических инструментов Graphical Toolbox поставляется в виде набора файлов с расширением jar. Для работы с Graphical Toolbox нужно установить файлы .jar на рабочей станции и настроить переменную среды CLASSPATH.

Убедитесь, что ваша рабочая станция соответствует всем требованиям, предъявляемым для работы IBM Toolbox for Java.

Установка Graphical Toolbox на рабочей станции

To develop Java programs using the Graphical Toolbox, first install the Graphical Toolbox JAR files on your workstation. Выберите один из следующих способов:

Передача файлов .jar

Примечание: Ниже описано несколько различных способов передачи файлов .jar. The IBM Toolbox for Java licensed program must be installed on your system. Кроме того, необходимо загрузить файл JAR для JavaHelp, jhall.jar, с Web-сайта Sun JavaHelp  .

- Передайте файлы .jar по FTP (убедитесь, что файлы передаются в двоичном режиме), скопировав их из каталога **/QIBM/ProdData/HTTP/Public/jt400/lib** в локальный каталог.
- Use System i Access for Windows to map a network drive.

Install JAR files with System i Access for Windows

You can also install the Graphical Toolbox when you install System i Access for Windows. The IBM Toolbox for Java is now shipped as part of System i Access for Windows. If you are installing System i Access for Windows for the first time, choose Custom Install and select the **IBM Toolbox for Java** component on the install menu. If you have already installed System i Access for Windows, you can use the Selective Setup program to install this component if it is not already present.

Настройка переменной CLASSPATH

Для работы с Graphical Toolbox его файлы .jar нужно указать в переменной среды CLASSPATH (либо задать их в опции `classpath` в командной строке).

Например, если вы скопировали файлы в каталог **C:\gtbox\lib** своей рабочей станции, добавьте следующие имена в переменную CLASSPATH:

```
C:\gtbox\lib\uitools.jar;
C:\gtbox\lib\jui400.jar;
C:\gtbox\lib\data400.jar;
C:\gtbox\lib\util400.jar;
C:\gtbox\lib\jhall.jar;
```

В переменную CLASSPATH необходимо также добавить каталог анализатора XML. Дополнительная информация приведена на следующей странице:

“Анализатор XML и обработчик XSLT” на стр. 418

If you have installed the Graphical Toolbox using System i Access for Windows, the JAR files (except jhall.jar) will all reside in the directory **\Program Files\Ibm\Client Access\jt400\lib** on the drive where you have installed System i Access for Windows. System i Access for Windows installs jhall.jar in the **\Program Files\Ibm\Client Access\jre\lib** directory. Соответствующие пути к файлам должны быть указаны в переменной CLASSPATH.

Описание файлов .jar

- **uitools.jar**: Contains the GUI Builder and Resource Script Converter tools.
- **jui400.jar**: Contains the runtime API for the Graphical Toolbox. Java programs use this API to display the panels constructed using the tools. Эти классы могут распространяться вместе с приложениями.
- **data400.jar**: Contains the runtime API for the Program Call Markup Language (PCML). Java programs use this API to call System i5 programs whose parameters and return values are identified using PCML. These classes may be redistributed with applications.
- **util400.jar**: Contains utility classes for formatting System i5 data and handling System i5 messages. Эти классы могут распространяться вместе с приложениями.

- **jhall.jar**: Contains the JavaHelp classes that displays the online help and context sensitive help for the panels you build with the GUI Builder.
- **XML parser**: Contains the XML parser used by the API classes to interpret PDML and PCML documents.

Примечание: Вы можете воспользоваться международными версиями инструментов GUI Builder и Resource Script Converter. Для запуска международной версии необходимо в процессе установки продукта Graphical Toolbox установить файл **uitools.jar**, соответствующий вашему языку, а также стране или региону. These JAR files are available on the server in **/QIBM/ProdData/HTTP/Public/jt400/Mri29xx**, where 29xx is the 4-digit i5/OS NLV code that corresponds to your language and country or region. The names of the JAR files in the various Mri29xx directories include 2-character suffixes for the Java language code and the country or region code. Этот дополнительный файл JAR помещается в переменную CLASSPATH перед файлом **uitools.jar**.

Работа с Graphical Toolbox

После установки Graphical Toolbox ознакомьтесь со следующими разделами, в которых приведена информация о работе с его компонентами:

- Работа с GUI Builder
- Работа с Resource Script Converter

Создание пользовательского интерфейса

Use the GUI Builder tool to create a user interface.

Для запуска GUI Builder введите следующую команду:

```
java com.ibm.as400.ui.tools.GUIBuilder [-plaf look and feel]
```

Если в переменной среды CLASSPATH не задан путь к файлам .jar пакета Graphical Toolbox, его нужно указать в командной строке с опцией `classpath`. См. раздел Настройка Graphical Toolbox.

Options `-plaf` *look and feel*

Задает внешний вид интерфейса на данной платформе. This option lets you override the default look and feel that is set based on the platform you are developing on, so you can preview your panels to see how they will look on different operating system platforms. The following look and feel values are accepted:

- Windows
- Metal
- Motif

В настоящий момент дополнительные атрибуты, применяемые в Swing 1.1, не поддерживаются GUI Builder

Типы ресурсов пользовательского интерфейса

При первом запуске GUI Builder необходимо создать новый файл PDML. From the menu bar on the GUI Builder widow, select **File --> New File**. После создания файла PDML вы можете определить для него любые из следующих типов ресурсов пользовательского интерфейса.

Панель

The fundamental resource type. It describes a rectangular area within which UI elements are arranged. The UI elements may consist of simple controls, such as radio buttons or text fields, images, animations, custom controls, or more sophisticated subpanels (see the following definitions for Split Pane, Deck Pane and Tabbed Pane). A panel may define the layout for a stand-alone window or dialog, or it may define one of the subpanels that is contained in another UI resource.

Меню Всплывающее окно со списком действий (например, ″Вырезать″, ″Скопировать″ и ″Вставить″). Для каждого действия можно определить клавиши быстрого доступа. В роли элемента меню может

выступать другое меню, переключатель или радиокнопка. Этот ресурс может задавать отдельное контекстное меню, выпадающее меню для одного из пунктов строки меню или саму строку меню для ресурса панели.

Панель инструментов
Окно с рядом кнопок, соответствующих действиям пользователя. На кнопке может быть размещен текст, изображение, либо и то, и другое. Панель инструментов можно сделать плавающей. В этом случае с ней можно будет работать как с отдельным окном, которое можно вынести за пределы панели.

Окно свойств
A stand-alone window or dialog consisting of a tabbed panels and OK, Cancel, and Help buttons. Panel resources define the layout of each tabbed window.

Мастер
A stand-alone window or dialog consisting of a series of panels that are displayed to the user in a predefined sequence, with Back, Next, Cancel, Finish, and Help buttons. The wizard window may also display a list of tasks to the left of the panels which track the user's progress through the wizard.

Разделенная панель
Субпанель, состоящая из двух панелей с разделителем между ними. Панели могут быть расположены горизонтально или вертикально.

Панель с закладками
Субпанель, представляющая управляющий элемент с закладками. Этот элемент может быть помещен на другую панель, разделенную панель или составную панель.

Составная панель
Субпанель, состоящая из нескольких панелей. В каждый момент времени на экране видна только одна панель. Составная панель может быть заменена на другую, например, в ответ на действие пользователя.

Таблица строк
Набор ресурсов строк и их идентификаторов.

Созданные файлы

Строки, которые должны быть преобразованы для панели, хранятся не в самом файле PDML, а в отдельном комплекте ресурсов Java. С помощью инструментов вы можете указать способ определения комплекта ресурсов: как файл PROPERTIES Java или как подкласс ListResourceBundle. Подкласс ListResourceBundle - это откомпилированная версия преобразуемых ресурсов. Его применение повышает производительность приложения на Java. Однако при этом замедляется процесс сохранения в GUI Builder, поскольку подкласс ListResourceBundle компилируется при каждой операции сохранения. Сначала рекомендуется создать файл PROPERTIES (опция по умолчанию). Позже, когда будет разработан окончательный вариант интерфейса, можно будет создать подкласс ListResourceBundle.

Для любой панели, описанной в файле PDML, можно создать шаблон справки в формате HTML. Если пользователь во время выполнения нажмет кнопку Справка или клавишу F1, то автоматически будет выдан раздел справки по текущему активному управляющему элементу. You must insert your help content at the appropriate points in the HTML, within the scope of the <!-- `HELPDOC:SEGMENTBEGIN` --> and <!-- `HELPDOC:SEGMENTEND` --> tags. For more specific help information see Editing Help Documents generated by GUI builder.

You can generate source code skeletons for the JavaBeans that will supply the data for a panel. Use the Properties window of the GUI Builder to fill in the DATACLASS and ATTRIBUTE properties for the controls which will contain data. The DATACLASS property identifies the class name of the bean, and the ATTRIBUTE property specifies the name of the gettor/settor methods that the bean class implements. Once you've added this information to the PDML file, you can use the GUI Builder to generate Java source code skeletons and compile them. At runtime, the appropriate gettor/settor methods will be called to fill in the data for the panel.

Примечание: The number and type of gettor/settor methods is dependent on the type of UI control with which the methods are associated. The method protocols for each control are documented in the class description for the DataBean class.

Finally, you can serialize the contents of your PDML file. Serialization produces a compact binary representation of all of the UI resources in the file. This greatly improves the performance of your user interface, because the PDML file must not be interpreted in order to display your panels.

To summarize: If you have created a PDML file named **MyPanels.pdml**, the following files will also be produced based on the options you have selected on the tools:

- **MyPanels.properties** - если вы запросили создание комплекта ресурсов в виде файла PROPERTIES
- **MyPanels.java** и **MyPanels.class** - если вы запросили создание комплекта ресурсов в виде подкласса ListResourceBundle
- **<panel name>.html** for each panel in the PDML file, if you have elected to generate online help skeletons
- **<dataclass name>.java** and **<dataclass name>.class** for each unique bean class that you have specified on your DATACLASS properties, if you have elected to generate source code skeletons for your JavaBeans
- **<resource name>.pdml.ser** for each UI resource defined in the PDML file, if you've elected to serialize its contents.

Примечание: Для применения функций условной обработки (SELECTED/DESELECTED) необходимо, чтобы имя панели не совпадало с именем панели, с которой связана условная функция. Например, если для ПАНЕЛИ1 из ФАЙЛА1 задано действие с предусловием, которое относится к полю ПАНЕЛИ1 из ФАЙЛА2, то система не сможет проверить это условие. Для устранения этой ситуации переименуйте ПАНЕЛЬ1 в ФАЙЛЕ2 и обновите событие условной обработки в ФАЙЛЕ1 соответствующим образом.

Запуск Resource Script Converter

Для запуска программы Resource Script Converter вызовите интерпретатор Java, введя следующую команду:

```
java com.ibm.as400.ui.tools.PDMLViewer
```

If you did not set your CLASSPATH environment variable to contain the Graphical Toolbox JAR files, then you will need to specify them on the command line using the `classpath` option. See Setting Up the Graphical Toolbox.

Для запуска программы Resource Script Converter в пакетном режиме введите следующую команду:

```
java
com.ibm.as400.ui.tools.RC2XML file [опции]
```

где *файл* - это имя обрабатываемого файла .rc. **Опции**

-x имя The name of the generated PDML file. Defaults to the name of the RC file to be processed.

-p имя The name of the generated PROPERTIES file. Defaults to the name of the PDML file.

-r имя The name of the generated ListResourceBundle subclass. Defaults to the name of the PDML file.

-package имя
The name of the package to which the generated resources will be assigned. If not specified, no package statements will be generated.

-l локаль
The locale in which to produce the generated resources. If a locale is specified, the appropriate 2-character ISO language and country or region codes are suffixed to the name of the generated resource bundle.

-h Создать шаблон электронной справки в формате HTML.

-d Создать шаблон исходного кода компонентов JavaBean.

-s Сохранить все ресурсы в двоичном виде.

Преобразование ресурсов Windows в формат PDML

All dialogs, menus, and string tables found in the RC file will be converted to the corresponding Graphical Toolbox resources in the generated PDML file. You can also define DATACLASS and ATTRIBUTE properties for Windows controls that will be propagated to the new PDML file by following a simple naming convention when you create the identifiers for your Windows resources. These properties will be used to generate source code skeletons for your JavaBeans when you run the conversion.

Идентификаторы ресурсов Windows задаются в следующем формате:

```
IDCB_<class name>_<attribute>
```

where `<class name>` is the fully-qualified name of the bean class that you want to designate as the DATACLASS property of the control, and `<attribute>` is the name of the bean property that you want to designate as the ATTRIBUTE property of the control.

For example, a Windows text field with the resource ID `IDCB_com_MyCompany_MyPackage_MyBean_SampleAttribute` produces a DATACLASS property of **com.MyCompany.MyPackage.MyBean** and an ATTRIBUTE property of **SampleAttribute**. If you elect to generate JavaBeans when you run the conversion, the Java source file **MyBean.java** is produced, containing the package statement **package com.MyCompany.MyPackage**, and gettor and settor methods for the **SampleAttribute** property.

Динамический просмотр панелей

Набор графических инструментов Graphical Toolbox содержит API, который может применяться в программах на Java для просмотра пользовательских панелей, определенных с помощью PDML. При выводе панелей этот API интерпретирует описание PDML и отображает элементы пользовательского интерфейса с помощью классов Java Foundation.

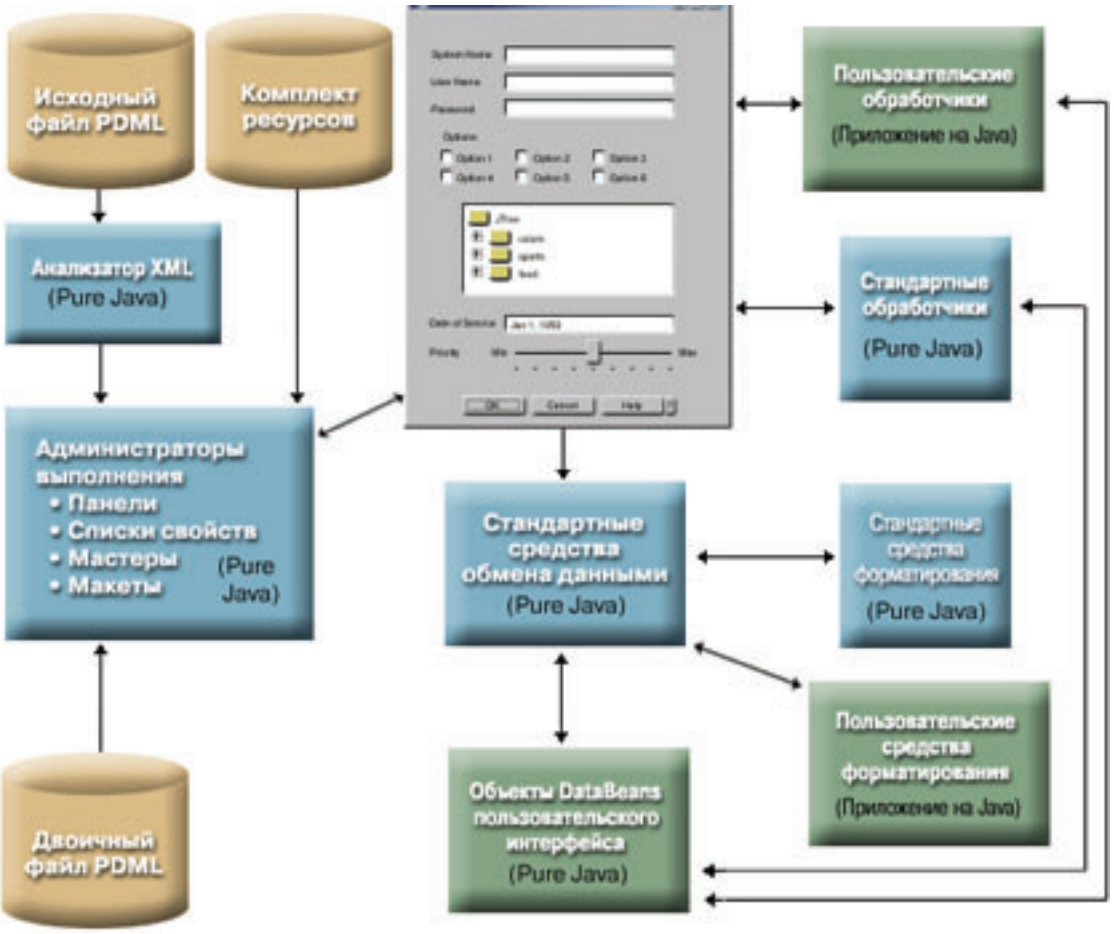
Среда выполнения Graphical Toolbox выполняет следующие задачи:

- Обрабатывает все данные, которыми обмениваются элементы управления пользовательского интерфейса и компоненты JavaBeans, описанные в PDML.
- Обеспечивает контроль над основными целочисленными и символьными типами в пользовательских данных и предоставляет интерфейс, позволяющий реализовать собственный контроль типов. При обнаружении ошибки в данных на экране появится сообщение об ошибке.
- Определяет стандартизованную обработку событий, связанных с фиксацией, отменой и вызовом справки, а также предоставляет средства обработки пользовательских событий.
- Управляет обменом данными между элементами управления пользовательского интерфейса на основе информации о состоянии, указанной в файле PDML. (Например, вы можете установить режим, в котором группа управляющих элементов становится недоступной при нажатии радиокнопки.)

The package com.ibm.as400.ui.framework.java contains the Graphical Toolbox runtime API.

The elements of the Graphical Toolbox runtime environment are shown in Figure 1. Программа на Java является клиентом одного или нескольких объектов, показанных в группе **Администраторы выполнения**.

Рис. 1: Среда выполнения Graphical Toolbox



Примеры

Предположим, что панель **MyPanel** определена в файле **TestPanels.pdml**, с которым связан файл свойств **TestPanels.properties**. Оба файла расположены в каталоге **com/ourCompany/ourPackage**, к которому можно обратиться из каталога, файла .zip или файла .jar, указанного в переменной CLASSPATH.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Пример: Создание и просмотр панели

Приведенная ниже программа создает и выводит панель:

```
import com.ibm.as400.ui.framework.java.*;

// Создание диспетчера панелей. Параметры:
// 1. Имя ресурса определения панели
// 2. Имя панели
// 3. Список компонентов DataBeans отсутствует

PanelManager pm = null;
try {
  pm = new PanelManager("com.ourCompany.ourPackage.TestPanels", "MyPanel", null);
}

catch (DisplayManagerException e) {
  e.displayUserMessage(null);
  System.exit(-1);
```

```
}

// Вывод панели
pm.setVisible(true);
```

Пример: Создание окна диалога

Если реализованы компоненты DataBean, содержащие данные для панели, и заданы соответствующие атрибуты в файле PDML, то для создания окна диалога может быть добавлен следующий код:

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

// Создание объектов, содержащих данные для панели
TestDataBean1 db1 = new TestDataBean1();
TestDataBean2 db2 = new TestDataBean2();

// Инициализация объектов
db1.load();
db2.load();

// Подготовка к передаче объектов в среду пользовательского интерфейса
DataBean[] dataBeans = { db1, db2 };

// Создание диспетчера панелей. Параметры:
// 1. Имя ресурса определения панели
// 2. Имя панели
// 3. Список компонентов DataBean
// 4. Владелец внешнего окна

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);
}

// Вывод панели
pm.setVisible(true);
```

Пример: Применение динамического диспетчера панелей

В диспетчере панелей появилась новая функция. Теперь он может динамически изменять размер панели. Рассмотрим тот же пример панели **MyPanel**, в котором используется динамический диспетчер панелей:

```
import com.ibm.as400.ui.framework.java.*;

// Создание динамического диспетчера панелей. Параметры:
// 1. Имя ресурса определения панели
// 2. Имя панели
// 3. Список компонентов DataBeans отсутствует

DynamicPanelManager dpm = null;
try {
  pm = new DynamicPanelManager("com.ourCompany.ourPackage.TestPanels", "MyPanel", null);
}

catch (DisplayManagerException e) {
  e.displayUserMessage(null);
  System.exit(-1);
```

```
}

// Вывод панели
pm.setVisible(true);
```

После запуска этого приложения появится панель, размер которой можно изменить. Поместите курсор на границу панели и, когда появится значок изменения размера, увеличьте или уменьшите размер панели.

Информация, связанная с данной

Package com.ibm.as400.ui.framework.java summary

Подробное описание рисунка 1: Среда выполнения Graphical Toolbox (rzahh504.gif)

found in IBM Toolbox for Java: Displaying your panels at runtime

На этом рисунке проиллюстрировано взаимодействие элементов среды выполнения Graphical Toolbox с кодом приложения.

Описание

На рисунке показано несколько элементов различного размера, формы и цвета, соединенных друг с другом однонаправленными и двунаправленными стрелками.

Для простоты поделим рисунок на три столбца и четыре строки, пронумеровав полученные области по порядку: слева направо, сверху вниз. Например, первая строка содержит области 1, 2 и 3; вторая строка - области 4, 5 и 6; и так далее.

- Окно диалога, расположенное в областях 2 и 5, представляет окно программы на Java. В этом окне находятся различные компоненты, в том числе переключатели, поля ввода и т.д.
- Два коричневых цилиндра, расположенных в верхней части области 1, называются Источник PDML и Комплект ресурсов. Эти цилиндры представляют источник PDML и файлы ресурсов Java, хранящиеся на носителе.
- Коричневый цилиндр в области 10 называется Двоичный код PDML. Он представляет двоичные файлы PDML, расположенные на носителе.
- Пять синих прямоугольников, окружающих нижнюю часть окна диалога, представляют компоненты продукта Graphical Toolbox. Ниже перечислены их имена, начиная с самого левого:
  - Анализатор XML (Pure Java) расположен в области 4. Он представляет Анализатор XML фирмы IBM.
  - Диспетчеры времени выполнения (Pure Java) расположены в области 7. Программа на Java является клиентом одного или нескольких объектов, содержащихся в этом прямоугольнике. Среди них есть панели, окна свойств, мастеры и макеты.
  - Общая система обмена данными (Pure Java) расположена в области 8.
  - Общие средства форматирования (Pure Java) расположены в области 9.
  - Общие обработчики (Pure Java) расположены в области 6.
- Три зеленых прямоугольника представляют исходный код, предоставленный разработчиком приложения. К ним относятся:
  - Пользовательские обработчики (Приложение на Java), расположенные в области 3
  - Пользовательские средства форматирования (Приложение на Java), расположенные в области 12
  - Компоненты данных JavaBean пользовательского интерфейса (Pure Java), расположенные в области 11
- Многие элементы рисунка соединены линиями:
  - Однонаправленная стрелка обозначает некоторое действие. Такая стрелка указывает на функцию или компонент, использующую объект, из которого исходит стрелка. Далее везде словосочетание ″компонент использует объект″ будет означать, что объект и компонент соединены однонаправленной стрелкой.

– Двунаправленная стрелка обозначает некоторое взаимодействие. Такая стрелка соединяет объекты, которые обмениваются информацией друг с другом. Далее везде словосочетание ″взаимодействие компонентов″ будет означать, что компоненты соединены двунаправленной стрелкой.

Графический интерфейс программы на Java (окно диалога, расположенное в областях 2 и 5) взаимодействует с Диспетчерами времени выполнения Graphical Toolbox (синий прямоугольник в области 7).

Диспетчеры времени выполнения (Pure Java) содержат панели, окна свойств, мастеры и макеты. Для создания графического интерфейса Диспетчеры времени выполнения применяют комплект ресурсов Java (один из двух коричневых цилиндров, расположенных в области 1) и данные PDML. Диспетчеры времени выполнения обрабатывают данные PDML одним из двух способов:

- Путем обработки двоичных файлов PDML (коричневый цилиндр в области 10)
- Using the IBM XML Parser (the blue rectangle in area 4), which in turn uses (parses) the PDML source files (one of two tan cylinders in area 1)

Программа на Java с графическим интерфейсом работает с данными одним из следующих способов:

- Графический интерфейс взаимодействует с пользовательскими обработчиками (зеленый прямоугольник в области 3) и общими обработчиками (синий прямоугольник в области 6)
- Общая система обмена данными (голубой прямоугольник в области 8) использует для получения информации графический интерфейс

Пользовательские обработчики, общие обработчики и общая система обмена данными взаимодействуют с компонентами данных JavaBean пользовательского интерфейса (зеленый прямоугольник в области 11), обмениваясь с ними информацией. Общая система обмена данными взаимодействует с общими средствами форматирования (синий прямоугольник в области 9) и пользовательскими средствами форматирования (зеленый прямоугольник в области 12) для преобразования данных в формат, поддерживаемый компонентами данных JavaBean.

Редактирование файлов справки, созданных с помощью GUI Builder

Для каждого файла проекта PDML GUI Builder создает шаблон справки и помещает его в отдельный документ HTML. Перед использованием этот документ HTML разбивается на отдельные файлы по темам для каждого окна диалога проекта PDML. Таким образом пользователь получает возможность работать со справкой по каждому разделу в отдельности, при этом общее число файлов справки остается небольшим.

Справочный документ является стандартным файлом HTML. Его можно просмотреть в любом браузере и изменить любым редактором HTML. Теги, определяющие разделы справочного документа, расположены внутри комментариев, поэтому они не видны в браузере. Теги комментария применяются для разбиения справочного документа на несколько разделов:

- Введение
- Раздел тем для каждого окна диалога
- Раздел тем для каждого управляющего элемента, для которого включена справка
- Заключение

Кроме этого, перед заключением вы можете вставить дополнительные разделы с общей информацией. До своего разбиения и вставки введения и заключения тематические разделы содержат только тело HTML. При разбиении справочного документа к каждому разделу добавляются введение и заключение, в результате чего образуются логически законченные файлы HTML. В качестве введения и заключения по умолчанию применяются аналогичные разделы из справочного документа, однако их можно заменить на собственный текст.

Структура справочного документа

Далее описываются составные части справочного документа:

Введение

Конец введения обозначается следующим тегом:

```
<!-- HELPDOC:HEADEREND -->
```

Если вы хотите заменить заголовок по умолчанию для всех или некоторых разделов справки, укажите ключевое слово HEADER и имя нужного файла HTML. Например:

```
<!-- HELPDOC:HEADEREND HEADER="defaultheader.html" -->
```

Раздел тем

Каждый тематический раздел обрамлен следующими тегами:

```
<!-- HELPDOC:SEGMENTBEGIN -->
```

и

```
<!-- HELPDOC:SEGMENTEND -->
```

Сразу после тега SEGMENTBEGIN следует тег с меткой, задающий имя раздела. В нем также указывается имя файла HTML, создаваемого при разбиении справочного документа. Имя раздела состоит из идентификатора панели, идентификатора управляющего элемента и расширения создаваемого файла (html). Например: MY_PANEL.MY_CONTROL.html. Для разделов, относящихся к панелям, указываются только идентификатор панели и расширение.

Программа создания справки добавит в справочный документ текст, указывающий, где следует разместить текст справки:

```
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" --><A NAME="MY_PANEL.MY_CONTROL.html"></A>
<H2>My favorite control</H2>
Вставьте сюда текст справки для объекта "Мой управляющий элемент".
<P><!-- HELPDOC:SEGMENTEND -->
```

Между меткой и тегом SEGMENTEND вы можете вставить произвольные теги HTML 2.0.

Тег PDMLSYNCH определяет, насколько тесно связаны раздел и управляющие элементы в PDML. Если PDMLSYCH имеет значение ″YES″, раздел справки будет удален при удалении управляющего элемента с таким же именем из документа PDML. В случае, если PDMLSYNCH=″NO″, раздел будет сохранен в документе справки независимо от наличия соответствующего управляющего элемента в PDML. Этот тег применяется, например, при создании дополнительных вложенных разделов или общего раздела.

Раздел справки для панели содержит ссылки на каждый справочный управляющий элемент панели. Эти ссылки содержат локальные справочные метки, позволяющие тестировать их как внутренние ссылки в стандартном браузере. При разбиении документа справки обработчик удаляет символ ″#″ из этих внутренних ссылок, превращая их во внешние ссылки итоговых файлов HTML, содержащих один раздел. Поскольку в некоторых случаях внутренние ссылки необходимо разместить в разделе, обработчик удаляет только начальные символы ″#″, если указатель содержит строку ″.html″.

Если вы хотите заменить заголовок по умолчанию для любого из разделов справки, укажите ключевое слово HEADER и имя нужного файла HTML. Например:

```
<!-- HELPDOC:SEGMENTBEGIN PDMLSYNCH="YES" HEADER="specialheader.html" -->
```

Заключение

Заключение справочного документа начинается со следующего тега:

```
<!-- HELPDOC:FOOTERBEGIN -->
```

The standard footer is </BODY></HTML>This footer is added to each HTML file.

Добавление ссылок

В справке можно использовать ссылки на любые внешние и внутренние URL, включая ссылки на другие разделы справки. При этом необходимо соблюдать следующие правила:

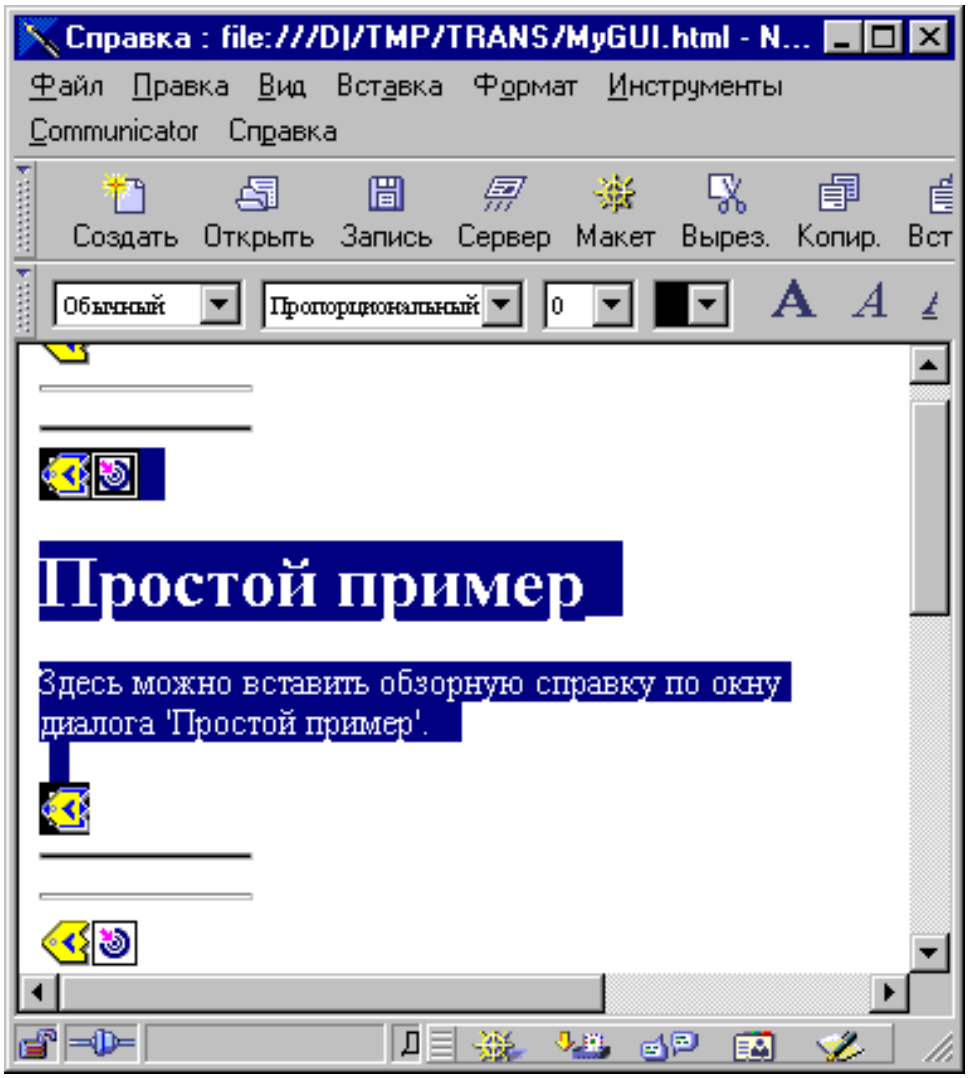
- Внешние URL используются стандартным образом. В них можно указывать внутренние составляющие ссылки.
- Ссылки внутри раздела являются стандартными, но не должны содержать расширения .html в имени тега. Это связано с тем, что программа обработки документа справки считает такие ссылки внешними, если разделы справки хранятся отдельно. Поэтому она удаляет начальный символ #.
- Ссылки на другие разделы следует указывать с начальным символом #, как если бы они были ссылками на внутренние метки документа.
- Можно также создавать внутренние ссылки на другие разделы. При обработке будут удалены только начальные символы ″#″.

Примечание:

- На этапе выполнения класс PanelManager выполняет поиск файлов справки в подкаталоге с тем же именем, что и у файла PDML. При разбиении справочного документа программа разработки по умолчанию создает такой каталог и помещает в него полученные файлы HTML.
- Программа обработки справочного файла не выполняет корректировки внешних ссылок, являющихся относительными ссылками. При использовании ссылке в отдельном файле справки относительные ссылки будут вычисляться начиная с нового подкаталога. Таким образом, вы должны будете скопировать внешние ресурсы (например, изображения) в нужный каталог или указать ″../″ в ссылках, чтобы поиск начинался с каталога панели.

Редактирование с помощью визуального редактора

Содержимое справки можно редактировать почти в любом графическом редакторе HTML. Поскольку теги HELPDOC являются комментариями, некоторые редакторы могут обрабатывать их неправильно. Для того чтобы обеспечить поддержку максимального числа редакторов, до и после тега SEGMENTBEGIN в структуре документа помещаются горизонтальные разделители. Благодаря этим разделителям документ в любом редакторе содержит четко выделенные разделы. При выборе сегмента для перемещения, копирования или удаления необходимо выбрать также оба горизонтальных разделителя, чтоб включить в выбранную область теги SEGMENTBEGIN. Горизонтальные разделители не копируются в итоговые отдельные файлы HTML.



Создание дополнительных разделов

В документе справки можно создавать дополнительные разделы. Чаще всего самый простой способ сделать это - скопировать другой раздел. При этом необходимо также скопировать горизонтальные разделители до и после тегов SEGMENTBEGIN. Это значительно упростит наглядное редактирование впоследствии и позволит избежать потери тегов. При редактировании файла справки следуйте приведенным ниже рекомендациям:

- Имя метки должно определять имя файла, в который будет помещен текст раздела при разбиении. Оно должно заканчиваться символами ″.html″.
- Для того чтобы раздел не был удален из структуры справки при повторном его создании, укажите ключевое слово PDMLSYNCH=″NO″ в теге SEGMENTBEGIN.
- Все ссылки на новый раздел будут внутренними ссылками, начинающимися с ″#″. Этот символ будет позже удален во время разбиения документа на отдельные файлы.

Проверка ссылок

В большинстве случаев вы можете проверить правильность ссылок путем загрузки документа в браузер и просмотра ссылок. В едином справочном документе ссылки хранятся во внутренней форме.

По окончании разработки, либо в том случае, если вы хотите проверить работу справки, вам понадобится разбить справочный документ на отдельные файлы. Для этого служит процедура Преобразование документа справки в HTML.

Если вам потребуется повторно создать справочный документ после редактирования, введенный ранее текст будет сохранен. Повторное создание документа может потребоваться при добавлении новых управляющих документов после создания шаблона справки. В этом случае программа создания справки проверит наличие документа перед тем, как создавать новый. Если она обнаружит справочный документ, то все существующие разделы будут сохранены, а в дополнение к ним будут добавлены разделы для новых управляющих элементов.

Работа с Graphical Toolbox в браузере

С помощью Graphical Toolbox вы можете создавать панели для аплетов Java, работающих в Web-браузере.

В этом разделе описано, как запустить в Web-браузере аплет, выводящий простую панель, который был описан в разделе Пример работы с Graphical Toolbox. Поддерживаются версии браузера не ниже Netscape 4.05 и Internet Explorer 4.0. Для того чтобы избежать необходимости учитывать особенности отображения аплетов в различных браузерах, рекомендуется применять встраиваемый модуль Sun Java. В противном случае вам потребуется создать для Netscape Navigator подписанные файлы .jar, а для Internet Explorer - подписанные файлы .cab.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Создание аплета

Исходный код аплета практически совпадает с исходным кодом рассмотренного ранее приложения на Java, однако он должен целиком располагаться в методе **init** подкласса **JApplet**. Кроме того, был добавлен фрагмент кода, в котором устанавливается размер панели, указанный в определении PDML. Ниже приведен исходный код примера аплета, хранящийся в файле **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
{
    // Выбор размера панели
    private PanelManager        m_pm;
    private Dimension           m_panelSize;

    // Определение исключительной ситуации на случай ошибки
    class SampleAppletException extends RuntimeException {}

    public void init()
    {
        System.out.println("Метод init!");

        // Трассировка параметров аплета
        System.out.println("SampleApplet code base=" + getCodeBase());
        System.out.println("SampleApplet document base=" + getDocumentBase());

        // Проверка виртуальной машины Java на совместимость со Swing 1.1
        if (System.getProperty("java.version").compareTo("1.1.5") < 0)
            throw new IllegalStateException("SampleApplet несовместим с версией JVM" +
                                            System.getProperty("java.version") +
                                              "- необходима версия 1.1.5 или выше");
// Создание объекта компонента, содержащего данные для панели
```

```
        SampleBean bean = new SampleBean();

        // Инициализация объекта
        bean.load();

        // Настройка DataBean для передачи компонента администратору панели
        DataBean[] beans = { bean };

        // Обновление строки состояния
        showStatus("Загрузка определения панели...");

        // Создание диспетчера панелей. Параметры:
        // 1. Имя файла PDML
        // 2. Имя панели
        // 3. Список объектов, содержащих данные для панели
        // 4. Панель аплета

        try { m_pm = new PanelManager("MyGUI", "PANEL_1", beans, getContentPane()); }
        catch (DisplayManagerException e)
        {
            // Произошла ошибка; вывод сообщения и выход из программы
            e.displayUserMessage(null);
            throw new SampleAppletException();
        }

        // Задание каталога, содержащего справку
        m_pm.setHelpPath("http://MyDomain/MyDirectory/");

        // Вывод панели
        m_pm.setVisible(true);
    }

    public void start()
    {
        System.out.println("Метод start!");

        // Установка заданных размеров панели
        m_panelSize = m_pm.getPreferredSize();
        if (m_panelSize != null)
        {
            System.out.println("Изменение размера на " +  m_panelSize);
            resize(m_panelSize);
        }
        else
            System.err.println("Ошибка: функция getPreferredSize возвратила пустое значение");
    }

    public void stop()
    {
        System.out.println("Метод stop!");
    }

    public void destroy()
    {
        System.out.println("Метод destroy!");
    }

    public void paint(Graphics g)
    {
        // Вызов родительского метода
        super.paint(g);

        // Сохранение исходного размера панели при изменении окна браузера
        if (m_panelSize != null)
            resize(m_panelSize);
    }
}
```

Панель содержимого аплета передается набору Graphical Toolbox. Она служит контейнером, который будет содержать создаваемую панель. Метод **start** устанавливает правильный размер панели аплета. Метод **paint** переопределяется, что позволяет сохранить размер панели при изменении размера окна браузера.

При работе с Graphical Toolbox в браузере файлы справки в формате HTML недоступны для файла jar. Они должны быть расположены в каталоге аплета в виде отдельных файлов. Имя этого каталога передается набору Graphical Toolbox посредством метода **PanelManager.setHelpPath**.

Теги HTML

Поскольку для создания среды выполнения Java нужного уровня рекомендуется применять встроенные модули Java фирмы Sun, код HTML для вызова аплета, применяющего набор Graphical Toolbox, будет достаточно сложным. Однако этот шаблон HTML, с небольшими модификациями, может применяться для вызова и других аплетов. Приведенный ниже текст правильно интерпретируется как браузером Netscape Navigator, так и браузером Internet Explorer. Если встроенный модуль Java не установлен на компьютере пользователя, то создается приглашение для его загрузки с Web-сайта фирмы Sun. Подробная информация о работе встраиваемого модуля Java приведена в документе Java Plug-in HTML Specification 

Ниже приведен код HTML примера аплета, содержащийся в файле **MyGUI.html**:

```
<html>

<head>
<title>Graphical Toolbox Demo</title>
</head>

<body>
<h1>Graphical Toolbox Demo Using Java(TM) Plug-in</h1>
<p>

<!-- BEGIN JAVA(TM) PLUG-IN APPLET TAGS -->

<!-- The following tags use a special syntax which allows both Netscape and Internet Explorer to load -->
<!-- встраиваемый модуль Java и запускать аплеты в среде JRE этого модуля. -->
<!-- Не изменяйте эти теги.           -->
<!-- For more information see http://java.sun.com/products/jfc/tsc/swingdoc-current/java_plug_in.html.-->

<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        width="400"
        height="200"
        align="left"
        codebase="http://java.sun.com/products/plugin/1.1.3/jinstall-113-win32.cab#Version=1,1,3,0">
    <PARAM name="code"     value="SampleApplet">
    <PARAM name="codebase" value="http://www.mycompany.com/~auser/applets/">
    <PARAM name="archive"  value="MyGUI.jar,jui400.jar,util400.jar,x4j400.jar">
    <PARAM name="type"     value="application/x-java-applet;version=1.1">

    <COMMENT>
    <EMBED type="application/x-java-applet;version=1.1"
           width="400"
           height=200"
           align="left"
           code="SampleApplet"
           codebase="http://www.mycompany.com/~auser/applets/"
           archive="MyGUI.jar,jui400.jar,util400.jar,x4j400.jar"
           pluginspage="http://java.sun.com/products/plugin/1.1.3/plugin-install.html">
        <NOEMBED>
    </COMMENT>
        Аплеты JDK 1.1 не поддерживаются.
        </NOEMBED>
    </EMBED>
</OBJECT>
```

```
<!-- END JAVA(TM) PLUG-IN APPLET TAGS -->

<p>
</body>
</html>
```

Версия должна быть 1.1.3.

Примечание: В этом примере файл JAR анализатора XML, **x4j400.jar**, хранится на Web-сервере. Вы можете использовать другие анализаторы XML. Дополнительная информация приведена в разделе “Анализатор XML и обработчик XSLT” на стр. 418. Это обязательно только в том случае, если файл PDML участвует в процедуре установки аплета. В целях повышения производительности рекомендуется преобразовывать определения панелей к *двоичному* формату, чтобы Graphical Toolbox не приходилось интерпретировать PDML во время выполнения. В результате преобразования создаются компактные двоичные представления панелей, что значительно повышает производительность пользовательского интерфейса. Дополнительная информация приведена в описании файлов, создаваемых сервисными средствами.

Установка и запуск аплета

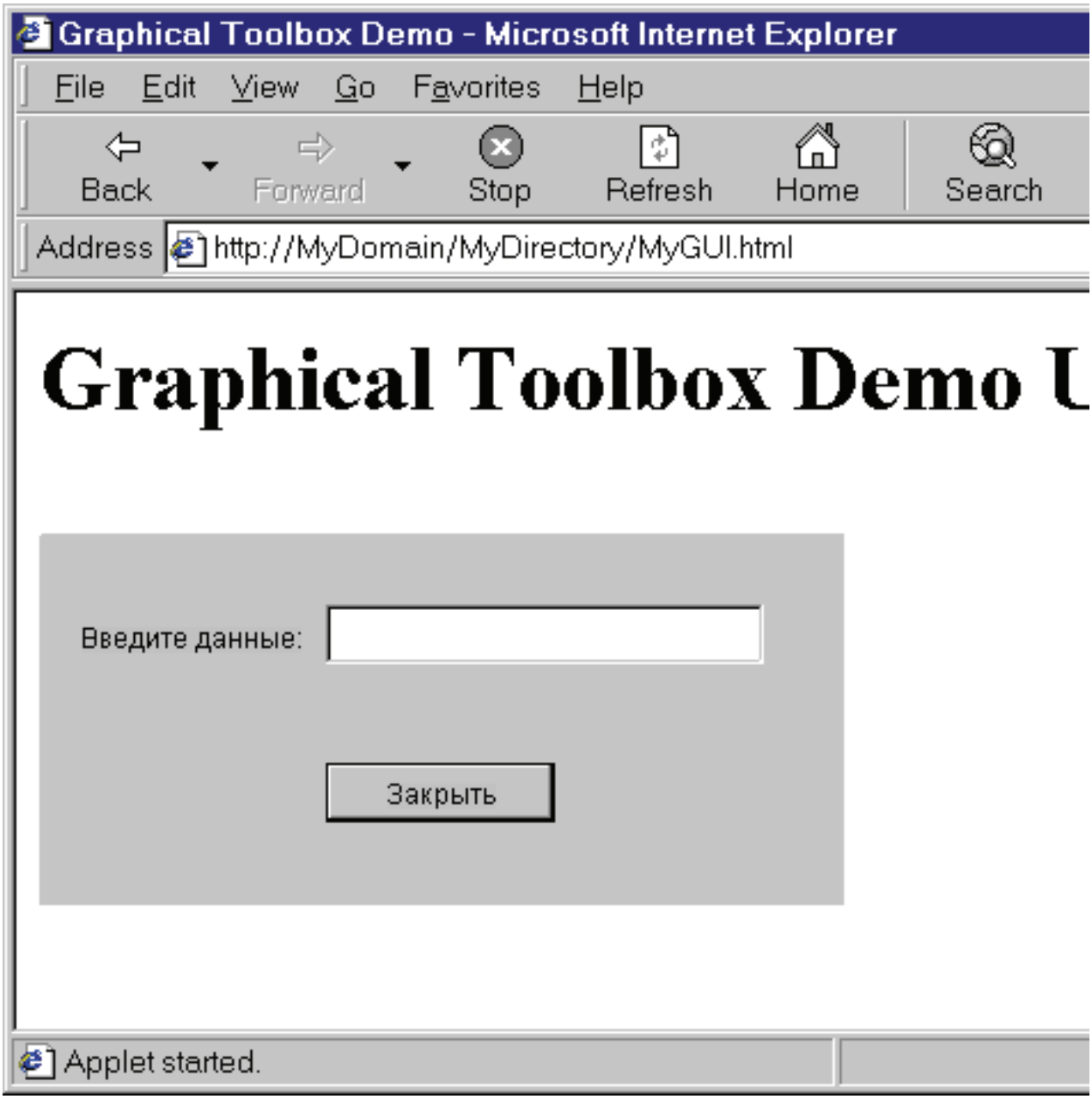
Установите аплет на выбранном Web-сервере, выполнив следующие действия:

1. Откомпилируйте файл **SampleApplet.java**.
2. Создайте файл .jar с именем **MyGUI.jar**, содержащий двоичное представление файлов аплета. К ним относятся файлы классов, созданные при компиляции файлов **SampleApplet.java** и **SampleBean.java**, файл PDML **MyGUI.pdml** и комплект ресурсов **MyGUI.properties**.
3. Скопируйте новый файл .jar в каталог на Web-сервере. Скопируйте файлы справки в формате HTML в каталог сервера.
4. Скопируйте файлы .jar набора Graphical Toolbox в каталог сервера.
5. Скопируйте файл **MyGUI.html**, содержащий встроенный аплет, в каталог сервера.

Совет: При тестировании аплетов убедитесь, что все файлы .jar Graphical Toolbox удалены из переменной среды CLASSPATH рабочей станции. В противном случае появится сообщение о том, что ресурсы аплета на сервере не найдены.

Теперь можно запустить аплет. Загрузите файл **MyGUI.html** с сервера в окно браузера. Если на компьютере еще не установлен встроенный модуль Java, появится приглашение для его установки. После установки встроенного модуля и запуска аплета окно браузера будет выглядеть примерно как на рис. 1:

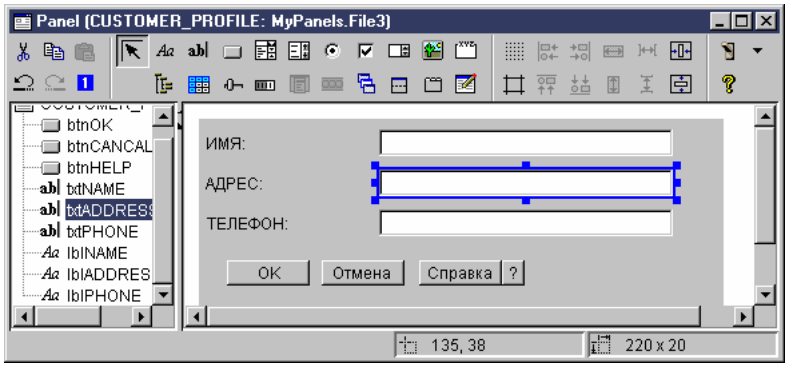
Рис. 1: Запуск примера аплета в браузере



Панель инструментов GUI Builder Panel Builder

На рис. 1 показано окно GUI Builder Panel Builder. Ниже рис. 1 приведено описание значков Panel Builder.

Рисунок 1: Окно GUI Builder Panel



Позволяет перемещать и изменять размеры компонентов панели.

Позволяет разместить на панели статическую метку.

Позволяет разместить в панели текст.

Позволяет добавить в панель кнопку.

Позволяет добавить в панель выпадающий список.

Позволяет добавить в панель список.

Позволяет добавить в панель радиокнопку.

Позволяет добавить в панель переключатель.

Позволяет добавить на панель поле прокрутки.

Позволяет добавить в панель изображение.

С его помощью можно разместить на панели меню.

С его помощью можно разместить на панели группу элементов.

С его помощью можно разместить на панели иерархическое дерево.

Позволяет добавить в панель таблицу.

Позволяет добавить в панель ползунок.

Позволяет добавить в панель индикатор состояния.

 Позволяет добавить в панель составную панель. Составная панель содержит несколько панелей. В каждый момент времени показана только одна панель, выбранная пользователем.

 С его помощью можно разместить на панели разделенную панель. Разделенная панель - это панель, разделенная на две части по вертикали или по горизонтали.

 С его помощью можно разместить на панели панель с закладками. Панель с закладками содержит несколько панелей. Для выбора любой из них пользователь должен щелкнуть на закладке. В закладке указан заголовок панели.

 Позволяет добавить на панель пользовательский компонент GUI.

 Позволяет разместить на панели панель инструментов.

 С его помощью можно разместить на панели сетку.

 Выравнивание нескольких компонентов панели по верхнему краю основного компонента.

 С его помощью можно выровнять несколько компонентов панели по нижнему краю основного или любого другого компонента.

 Позволяет выровнять высоту нескольких компонентов по высоте основного или любого другого компонента.

 Позволяет разместить выбранный компонент в центре панели по вертикали.

 С его помощью можно просмотреть поля панели.

 Выравнивание нескольких компонентов панели по левому краю основного компонента.

 Позволяет выровнять несколько компонентов по левому краю основного или другого выбранного компонента.

 Позволяет придать нескольким компонентам ширину основного или другого выбранного компонента.

 Горизонтальное выравнивание выбранных компонентов по центру панели.

 Позволяет вырезать компоненты панели.

 Позволяет скопировать компоненты в буфер обмена.

 Позволяет вставить компоненты из буфера обмена.

 Отмена последнего действия.

 Повтор последнего действия.

 Изменяет порядок перехода по элементам при нажатии клавиши TAB.

 Показывает окончательный вид созданной панели.

 С его помощью можно просмотреть дополнительную информацию о Graphical Toolbox.

Объекты Javabean IBM Toolbox for Java

Объекты JavaBean представляют собой написанные на языке Java программные компоненты, которые могут многократно использоваться в различных приложениях. Компонент - это логически законченный программный модуль, который может описывать как метку или кнопку, так и целое приложение.

Объекты JavaBean могут быть визуальными или невизуальными. Невизуальные JavaBean всегда имеют визуальное представление (значок или имя), позволяющее работать с ними в визуальных средах.

Многие общие классы IBM Toolbox for Java также являются компонентами JavaBean. Эти классы соответствуют стандартам JavaBean Javasoft и могут использоваться многократно. Свойства и методы компонента JavaBean IBM Toolbox for Java Java совпадают со свойствами и методами класса.

Объекты JavaBean могут использоваться в прикладной программе и в средствах визуального программирования, таких как IBM VisualAge for Java.

Примеры

Ниже приведен пример применения JavaBeans в программе и создания программы на основе JavaBeans с помощью визуального компоновщика:

“Пример: Код компонента IBM Toolbox for Java” на стр. 560

“Пример: Создание объектов JavaBean с помощью визуального компоновщика” на стр. 561

JDBC

JDBC - это интерфейс прикладных программ (API), который входит в пакет Java и позволяет программам на Java работать с широким спектром баз данных.

Драйвер JDBC IBM Toolbox for Java предоставляет API для вызова операторов языка структурных запросов (SQL) и обработки информации, полученной из базы данных сервера. You can also use IBM Developer Kit for Java JDBC driver, called the ’native’ JDBC driver:

- Драйвер JDBC IBM Toolbox рекомендуется применять в том случае, если программа на Java и файлы базы данных расположены в разных системах, как, например, в среде клиент-сервер
- R Use the native JDBC driver when both the Java program and database files are on the same server

Различные версии JDBC

Существует несколько версий API JDBC. Драйвер JDBC IBM Toolbox for Java поддерживает следующие версии:

- API JDBC 1.2 (пакет java.sql) входит в состав базовых API продуктов Java Platform 1.1 и JDK 1.1.

- Базовый API JDBC 2.1 (пакет java.sql) входит в пакет Java 2 Platform, Standard Edition (J2SE) и Java 2 Platform Enterprise Edition (J2EE).
- API из дополнительного пакета JDBC 2.0 (пакета javax.sql) входят в состав продукта J2EE. Его можно загрузить с Web-сайта фирмы Sun.Раньше дополнительный пакет назывался стандартным расширением JDBC 2.0.

@ - JDBC 3.0 API (the java.sql and javax.sql packages) is included in J2SE, Version 1.4 and 5.0.

+ - JDBC 4.0 API is included in Java SE Version 6.

+ **Примечание:** Support for the JDBC 4.0 API is available only in JTOpen. It is not available in IBM Toolbox for Java, 5761-JC1.

+ Enhancements to IBM Toolbox for Java JDBC support for V6R1

+ Many additions were made to JDBC support in V6R1.

+ The enhancements to IBM Toolbox for Java JDBC support are detailed in the following sections:

+ - "Support for JDBC 4.0"
+ - "Query storage limit"
+ - "Decimal float (DECFLOAT) data type" на стр. 321
+ - "Passing the client type and application name to the server" на стр. 321
+ - "Maximum length of cursor names extended" на стр. 322
+ - "Generated key support" на стр. 322
+ - "Improved default value support" на стр. 323
+ - "Increased maximum in GROUP BY clause" на стр. 323
+ - "Batch update support" на стр. 323

+ Support for JDBC 4.0

+ Support for the JDBC 4.0 API with Java SE Version 6 is available only in the open source JTOpen version of IBM Toolbox for Java. It is not available in IBM Toolbox for Java, 5761-JC1.

+ Query storage limit

+ You can use the query storage limit property to limit the storage used by a query. This property compares the storage limit you specify to the estimated storage usage of the query. If the estimated storage usage exceeds the specified storage limit, the query is not allowed to run.

+ Beginning in V6R1, the IBM Toolbox for Java JDBC driver will add the following methods to the AS400JDBCDataSource class:

```
+         setQueryStorageLimit()
+         public void setQueryStorageLimit(int limit);
```

+ Specifies the query storage limit value to be used when statements in a connection are executed. Valid values are -1 to 2 147 352 578. The default is value -1, indicating a special value of *NOMAX.

```
+         getQueryStorageLimit()
+         public int getQueryStorageLimit()
```

+ Returns the query storage limit value used when statements in a connection are executed. The default value is -1, indicating a special value of *NOMAX.

+ Systems that run a release of i5/OS prior to V6R1 will ignore the getQueryStorageLimit property.

Decimal float (DECFLOAT) data type

The decimal float (DECFLOAT) inherits favorable properties from both float and decimal data types. Data values of DECFLOAT are stored such that trailing zeros are significant: for example, 2.0 and 2.00 are different binary representations. In SQL comparisons, however, the values are treated as equal.

The IBM Toolbox for Java JDBC driver has added the following methods to the AS400JDBCDataSource class:

```
setDecfloatRoundingMode()
public void setDecfloatRoundingMode(int String mode)
```

Specifies the rounding mode to be used for DECFLOAT numbers.

```
getDecfloatRoundingMode()
public intString getDecfloatRoundingMode()
```

Returns the rounding mode to be used for DECFLOAT numbers.

Valid values for these methods include:

half even (default value)
Round to nearest digit. If equidistant between two digits, round to the nearest even digit. The numeric value of this constant is 0.

half up Round to nearest digit. If equidistant between two digits, round up. The numeric value of this constant is 1.

down Round to nearest lower digit. The numeric value of this constant is 2.

ceiling Round towards positive infinity. The numeric value of this constant is 3.

floor Round towards negative infinity. The numeric value of this constant is 4.

half down
Round to the nearest digit. If equidistant between two digits, round down. The numeric value of this constant is 5.

up Round to nearest higher digit. The numeric value of this constant is 6.

Passing the client type and application name to the server

Web applications need a way to pass end-user-client information to the database server so that more detailed information may be logged. The IBM Toolbox for Java JDBC driver allows an application to override this information by calling the following java.sql.Connection.setClientInfo() methods:

```
void AS400JDBCConnection.setClientInfo(java.lang.String name,java.lang.String value)
```

This method sets the value of the client info property specified by name to the value specified by `value`. See the DatabaseMetadata.getClientInfoProperties method description for the client info properties supported by the IBM Toolbox for Java JDBC driver.

```
void AS400JDBCConnection.setClientInfo(java.util.Properties properties)
```

This method sets the value of the connection's client info properties. The Properties object contains the names and values of the client info properties to be set. The set of client info properties contained in the properties list replaces the current set of client info properties on the connection. If a property that is currently set on the connection is not present in the properties list, that property is cleared. Specifying an empty properties list will clear all of the properties on the connection.

```
String AS400JDBCConnection.getClientInfo(java.lang.String name)
```

+ This method returns the value of the client info property specified by `name`. This method may return `null` if the
+ specified client info property has not been set and does not have a default value.

+ `Properties AS400JDBCConnection.getClientInfo()`

+ This method returns a list containing the name and current value of each client info property supported by the driver.
+ The value of a client info property may be null if the property has not been set and does not have a default value.

+ `ResultSet AS400JDBCDatabaseMetaData.getClientInfoProperties()`

+ This method retrieves a list of the client info properties that the driver supports. The IBM Toolbox for Java JDBC
+ driver returns a result set with the following information:

+ *Таблица 2. Result set for getClientInfoProperties*

Name	Максимальная длина	Значение по умолчанию	Description
ApplicationName	255	""	The name of the application currently utilizing the connection.
ClientAccounting	255	""	Accounting information.
ClientHostname	255	""	The hostname of the computer the application using the connection is running on.
ClientProgramID	255	""	The client program identification.
ClientUser	255	""	The name of the user that the application using the connection is performing work for. This may not be the same as the user name that was used in establishing the connection.

+ Maximum length of cursor names extended

+ Beginning in V6R1, the maximum length of cursors will be 128 characters. The previous maximum length was 8
+ characters. An application can set the name of a cursor by calling the java.sql.Statement.setCursorName() method.

+ Generated key support

+ In past releases, only one row of information could be returned from a multiple row insert operation. Beginning in
+ V6R1, you will be able to access more information about a multiple row insert operation. This will allow you to
+ retrieve generated column information such as ROWID, identity column, sequence, or generated expressions.
+ Generated key support will be implemented in the following IBM Toolbox for Java JDBC methods:
+ • AS400JDBCStatement.getGeneratedKeys()
+ • AS400JDBCStatement.execute(String sql, int autoGeneratedKeys)
+ • AS400JDBCStatement.execute (String sql, int[] columnIndexes)
+ • AS400JDBCStatement.execute (String sql, String[] columnNames)
+ • AS400JDBCStatement.executeUpdate(String sql, int[] autoGeneratedKeys)
+ • AS400JDBCStatement.executeUpdate (String sql, int[] columnIndexes)
+ • AS400JDBCStatement.executeUpdate (String sql, String[] columnNames)
+ • AS400JDBCConnection.prepareStatement(String sql, int autoGeneratedKeys)
+ • AS400JDBCConnection.prepareStatement(String sql, int[] columnIndexes)
+ • AS400JDBCConnection.prepareStatement(String sql, String[] columnNames)

+ Improved default value support

+ Beginning in V6R1, The IBM Toolbox for Java JDBC driver will return a column's default value as a string through
+ the DatabaseMetaData.getColumns() method. If the default value is null, a string with the value 'NULL' will be
+ returned.

+ Increased maximum in GROUP BY clause

+ The new value for DatabaseMetaData getMaxColumnsInGroupBy( ) is 8000.

+ Batch update support

+ The improved batch update support in V6R1 will provide better information to you when you are running a
+ multiple-row batched insert statement. There will be a field in the diagnostics area and SQLCA that will contain the
+ count of successful statements so that you can better determine the location of any errors.

+ The IBM Toolbox for Java JDBC driver will use this information when creating a java.sql.BatchUpdateException.
+ AS400JDBCStatement and AS400JDBCPreparedStatement will also use this information to return the correct
+ information back from the executeBatch() method.

Изменения, внесенные в поддержку JDBC продукта IBM Toolbox for Java в версии V5R4

Часть функций JDBC была расширена в i5/OS версии 5 выпуска 4.

Далее перечислены функции JDBC, расширенные в i5/OS версии 5 выпуска 4:

- "Размер операторов - 2 Мб "
- "Поддержка имен столбцов длиной 128 байт "
- "Поддержка трассировки на сервере базы данных " на стр. 324
- "eWLM Correlator " на стр. 324

Информация о расширенных функциях JDBC для предыдущих выпусков системы приведена в разделах "Расширенная поддержка JDBC в версии 5 выпуска 3" на стр. 324 и "Расширенные функции JDBC в i5/OS версии 5 выпуска 2" на стр. 325.

Размер операторов - 2 Мб

Prior to V5R4, the limit on SQL statement size was 65 535 bytes. This corresponds to 65 535 characters when the statement text is represented using a single-byte CCSID, and 32 767 characters when the statement text is represented using a double-byte CCSID. Это ограничение влияло на некоторых пользователей, особенно тех, кто работал с автоматически генерируемыми операторами SQL.

R In V5R4, the System i statement size limit has been increased to two megabytes, or 2 097 152 bytes. Драйвер JDBC
R IBM Toolbox for Java всегда отправляет текст операторов в двухбайтовой кодировке Unicode. Therefore, the
R maximum statement length in characters will be one megabyte or 1 048 576 characters.

Поддержка имен столбцов длиной 128 байт

В версии V5R4 база данных поддерживает названия столбцов SQL длиной до 128. До V5R4 поддерживались имена длиной до 30 байт. Драйвер JDBC IBM Toolbox for Java поддерживает работу с такими длинными именами.

В поддержке 128-байтовых имен столбцов есть одно исключение. Если используется локальное кэширование пакетов, и имена столбцов превышают 30 символов, то сервер возвратит имена столбцов как системные имена.

Поддержка трассировки на сервере базы данных

В драйвер JDBC Toolbox for Java добавлена поддержка трассировки на сервере. Эта функция включается опцией ″64″ свойства соединений ″трассировка сервера″. Дополнительная информация приведена в разделе “IBM Toolbox for Java JDBC properties” на стр. 328.

eWLM Correlator

IBM Toolbox for Java может работать с IBM Enterprise Workload Manager (eWLM) Correlator и передавать его хосту как фактор атрибута соединения для работы с ARM. Этот атрибут может быть отправлен хосту в любое время после установки соединения. Для этого служит один из методов класса AS400JDBCConnection:

setDB2eWLMCorrelator

```
public void setDB2eWLMCorrelator(byte[] bytes)
                         throws SQLException
```

Задает eWLM Correlator. Предполагается, что будет задано допустимое значение. Если значение равно null, реализация ARM/eWLM будет выключена. Корреляторы eWLM могут работать с i5/OS V5R3 или более поздней. This request is ignored when running to V5R2 or earlier servers.

Параметры:

- `bytes`: значение eWLM Correlator
- `SQLException`: См. информацию на Web-сайте Sun Microsystems, Inc.: Class SQLException .

Информация о расширенных функциях JDBC для предыдущих выпусков системы приведена в разделах “Расширенная поддержка JDBC в версии 5 выпуска 3” и “Расширенные функции JDBC в i5/OS версии 5 выпуска 2” на стр. 325.

Расширенная поддержка JDBC в версии 5 выпуска 3

Several JDBC functions were enhanced for i5/OS Version 5 Release 3.

Enhanced JDBC functions for i5/OS Version 5 Release 3 include:

- Поддержка UTF-8 и UTF-16
- Поддержка типов данных Binary и Varbinary
- Десятичные вычисления повышенной точности
- Поддержка больших объектов (LOB) размером до 2 Гб:
- Поддержка курсоров без учета изменений
- Поддержка таблиц материализованных запросов

For information about enhanced JDBC functions for previous releases, see V5R2 enhancements to IBM Toolbox for Java JDBC support.

Поддержка UTF-8 и UTF-16

Данные UTF-8 хранятся в символьных полях с CCSID, равным 1208. Некомбинируемый символ UTF-8 - это набор байтов переменной длины (один, два, три или четыре), а комбинируемый символ - это последовательность байтов любой длины. Длина символьного поля - это максимальное число байтов, которые оно может содержать. С помощью CCSID 1208 UTF-8 можно задать данные следующих типов:

- Символы фиксированной длины (CHAR)
- Символы переменной длины (VARCHAR)
- Символы LOB (CLOB)

Данные UTF-16 хранятся в графических полях с CCSID 1200. Некомбинируемый символ UTF-8 может состоять из одного или двух байтов (искусственные символы), а комбинируемый символ может иметь неограниченную длину. Длина графического поля - это максимальное число байтов, которые оно может содержать. С помощью CCSID 1208 UTF-8 можно задать данные следующих типов:

- Символы фиксированной длины (CHAR)
- Символы переменной длины (VARCHAR)
- Двухбайтовые символы LOB (DBCLOB)

Поддержка типов данных Binary и Varbinary

Типы данных BINARY и VARBINARY схожи с типами CHAR и VARCHAR, за исключением того, что они содержат двоичные, а не символьные данные. Поля BINARY имеют фиксированную длину. Поля VARBINARY могут быть разной длины. Типы данных BINARY и VARBINARY имеют следующие характеристики:

- Идентификатор набора символов (CCSID) для всех двоичных типов равен 65535
- В операторах присваивания и сравнения двоичные переменные совместимы только с переменными других двоичных типов данных (BINARY, VARBINARY и BLOB)
- Символом заполнения для двоичных данных является не пробел, а x'00'
- В случае, если для предотвращения ошибок усечения конечные символы необходимо удалить, вместо пробелов удаляются символы x'00'
- При сравнении двоичных данных они будут равны, только если совпадает их содержимое и размер. При сравнении конечные нули не игнорируются
- Если при сравнении двух полей одно из них длиннее другого, то более короткое поле считается меньшим, если все остальные символы полей совпадают

Десятичные вычисления повышенной точности

В этой версии десятичные вычисления выполняются с точностью 63 разряда. Появились также три новых свойства - минимальная длина дробной части, максимальная точность и максимальная длина значения; кроме того, добавлены шесть новых методов класса AS400JDBCDataSource: setMinimumDivideScale(int divideScale), getMinimumDivideScale(), setMaximumPrecision(int precision), getMaximumPrecision(), setMaximumScale(int scale) и getMaximumScale(). Минимальная длина дробной части соответствует минимальной дробной части при делении и может принимать значения от 0 до 9. Максимальная точность - это максимальное число десятичных знаков после запятой, она может быть равна 31 или 63. Максимальная длина значения - это наибольшее возможное число знаков числа, она может принимать значения от 0 до 63.

Поддержка больших объектов (LOB) размером до 2 Гб:

Enhancements for IBM Toolbox for Java JDBC now allow the use of up to 2 GB LOBs support.

Поддержка курсоров без учета изменений

В этой версии поддерживаются курсоры, работающие без учета изменений. Они используются, если в объекте ResultSet задан параметр TYPE_SCROLL_INSENSITIVE. Объект ResultSet не передает изменения в основную базу данных, если она открыта.

Поддержка таблиц материализованных запросов

Возвращает значение ″MATERIALIZED QUERY TABLE″ параметра TABLE_TYPE при вызове метода DatabaseMetaData.getTables().

Расширенные функции JDBC в i5/OS версии 5 выпуска 2

Часть функций JDBC была расширена в i5/OS версии 5 выпуска 2.

Далее перечислены функции JDBC, расширенные в i5/OS версии 5 выпуска 2:

- Удалено ограничение 'FOR UPDATE'
- Изменение функции усечения данных
- Получение и изменение столбцов и параметров по имени

- Получение автоматически создаваемых ключей
- Повышение производительности при обработке операторов вставки SQL в пакетном режиме
- Расширенная поддержка метода ResultSet.getRow()
- Изменены правила использования символов разных регистров в именах столбцов
- Возможность настройки уровня блокировки для объектов Statement, CallableStatement и PreparedStatement
- Расширенная поддержка уровня изоляции транзакций

Удалено ограничение 'FOR UPDATE'

Для создания курсора с возможностью обновления в операторах SELECT больше не нужно указывать предложение FOR UPDATE. При подключении к системам i5/OS версии V5R1 или выше IBM Toolbox for Java обрабатывает уровень распараллеливания, заданный при создании оператора. Если уровень распараллеливания не задан, то по умолчанию применяется курсор, допускающий только чтение.

При усечении данных исключительные ситуации создаются только в том случае, если усеченные символьные данные сохраняются в базе данных

Data truncation rules for IBM Toolbox for Java now are the same as those for the IBM Developer Kit for Java JDBC driver. Дополнительная информация приведена в разделе IBM Toolbox for Java - Свойства JDBC.

Получение и изменение столбцов и параметров по имени

Новые методы позволяют по имени столбца получать и обновлять информацию в объекте ResultSet, а также получать и задавать информацию по имени параметра в объекте CallableStatement. Например, если ранее для объекта ResultSet были вызваны следующие операторы:

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

То теперь можно задать следующий оператор:

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

Обратите внимание, что обращение к параметрам с помощью индекса занимает меньше времени, чем обращение по имени. Кроме того, с помощью объекта CallableStatement можно задать имена параметров. Так, если ранее для объекта CallableStatement был вызван следующий оператор:

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

То теперь можно задать следующий оператор:

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

Для применения этих методов необходимы продукты JDBC версии 3.0 или выше и Java 2 Platform версии 1.4 (Standard или Enterprise Edition).

Получение автоматически создаваемых ключей

Метод getGeneratedKeys() класса AS400JDBCStatement получает информацию обо всех ключах, автоматически созданных в результате выполнения объекта Statement.Если объект Statement не создал ни одного ключа, в качестве результата будет возвращен пустой объект ResultSet. На данный момент сервер позволяет получить информацию только об одном автоматически созданном ключе (ключе последней вставленной строки). В следующем примере в таблицу вставляется значение, а затем считывается автоматически созданный ключ:

```
R       Statement s =
R          statement.executeQuery("INSERT INTO MYSCHOOL/MYSTUDENTS (FIRSTNAME) VALUES ('JOHN'");
R       ResultSet rs = s.getGeneratedKeys();
R            // Currently the server supports returning only one auto-generated
R            // ключ -- ключ последней вставленной строки.
R       rs.next ();
```

```
R               String autoGeneratedKey = rs.getString(1);
R                    // Автоматически созданный ключ может быть задан, например,
R                    // в качестве первичного ключа другой таблицы
```

Для получения автоматически создаваемых ключей необходимы продукты JDBC версии 3.0 или выше и Java 2 Platform версии 1.4 (Standard или Enterprise Edition). Кроме того, необходимо, чтобы соединение было установлено с системой i5/OS версии V5R2 или выше.

Повышение производительности при выполнении операторов вставки SQL в пакетном режиме

Теперь операторы вставки SQL в пакетном режиме обрабатываются быстрее. Для обработки операторов SQL в пакетном режиме применяются различные методы addBatch(), предусмотренные в классах AS400JDBCStatement, AS400JDBCPreparedStatement и AS400JDBCCallableStatement.Указанное изменение касается только операторов вставки. При обработке нескольких операторов вставки в пакетном режиме потребуется обратиться к серверу только один раз. Однако при обработке операторов вставки, обновления и удаления в пакетном режиме каждый запрос будет передан на сервер по-отдельности.

Для применения пакетного режима необходимы продукты JDBC версии 2.0 или выше и Java 2 Platform версии 1.2 (Standard или Enterprise Edition).

Расширенная поддержка метода ResultSet.getRow()

В предыдущих выпусках драйвер JDBC продукта IBM Toolbox for Java поддерживал метод getRow() объекта ResultSet лишь частично. В частности, при вызове методов ResultSet.last(), ResultSet.afterLast() и ResultSet.absolute() с отрицательным параметром номер текущей строки становился недоступным. В новой версии эти ограничения устранены.

Использование символов различных регистров в именах столбцов

Методы IBM Toolbox for Java сравнивают имена столбцов пользователя или приложения с именами столбцов в базе данных. Если имя столбца не заключено в кавычки, IBM Toolbox for Java преобразует все буквы имени в прописные, а затем сравнивает его с именем, хранящимся на сервере. Если имя столбца заключено в кавычки, то оно должно в точности совпадать с именем, хранящимся на сервере. В противном случае IBM Toolbox for Java генерирует исключительную ситуацию.

Задание уровня блокировки при создании объектов Statement, CallableStatement и PreparedStatement

Новые методы класса AS400JDBCConnection позволяют задавать уровень блокировки для создаваемых объектов Statement, CallableStatement и PreparedStatement.Уровень блокировки указывает, остается ли курсор открытым при фиксации транзакции. Теперь уровень блокировки оператора может отличаться от уровня блокировки объекта соединения. Кроме того, с объектом соединения может быть связано несколько операторов открытия, для каждого из которых задан свой уровень блокировки. При выполнении фиксации каждый объект обрабатывается в соответствии с указанным уровнем блокировки.

Ниже указан порядок наследования уровня блокировки:

1. Уровень блокировки, указанный при создании оператора с помощью метода класса Connection (createStatement(), prepareCall() или prepareStatement()).
2. Уровень блокировки, указанный с помощью метода Connection.setHoldability(int).
3. Holdability specified by the IBM Toolbox for Java JDBC cursor hold property (when methods in 1. or 2. are not used)

Для применения этих методов необходимы продукты JDBC версии 3.0 или выше и Java 2 Platform версии 1.4 (Standard или Enterprise Edition). На серверах с операционной системой i5/OS версии V5R1 или ниже применяется только уровень блокировки, указанный в свойстве JDBC.

Расширенная поддержка уровня изоляции транзакций

Драйвер JDBC продукта IBM Toolbox for Java теперь позволяет изменить уровень изоляции транзакций на *NONE после установления соединения. В версиях ниже V5R2 драйвер JDBC IBM Toolbox for Java генерировал исключительную ситуацию при изменении значения на *NONE после создания соединения.

IBM Toolbox for Java JDBC properties

При подключении к базе данных сервер с помощью драйвера JDBC могут быть заданы различные свойства. Все свойства необязательны. Их можно указать в URL подключения или в объекте java.util.Properties. Если свойство задано и в URL, и объекте Properties, будет использоваться значение, указанное в URL.

Примечание: Приведенный ниже список не содержит свойств класса DataSource.

В приведенной ниже таблице перечислены свойства соединений, поддерживаемые драйвером. Некоторые из них влияют на производительность, другие управляют атрибутами задания сервера. Различные свойства разбиты на следующие категории:

Общие свойства

Общие свойства - это системные атрибуты, задающие имя и пароль пользователя и указывающие, нужно ли выводить приглашение подключения к серверу.

Property	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″password″	Указывает пароль для подключения к серверу. Если пароль не указан, он будет запрошен у пользователя, при условии, что значение свойства password не равно false (в противном случае соединение установлено не будет).	нет	системный пароль	(запросить у пользователя)
″prompt″	Указывает, будет ли выдаваться приглашение для ввода имени и пароля пользователя, если их необходимо указать для подключения к серверу. Если для подключения к системе требуется запросить пароль у пользователя и это свойство равно false, то соединение не устанавливается.	нет	″true″ ″false″	″true″
″user″	Указывает имя пользователя для подключения к серверу. Если пароль не указан, он будет запрошен у пользователя, при условии, что значение свойства password не равно false (в противном случае соединение установлено не будет).	нет	имя пользователя сервера	(запросить у пользователя)

Свойства сервера

Свойства сервера - это атрибуты, управляющие транзакциями, библиотеками и базами данных.

Property	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″cursor hold″	Указывает, будет ли блокироваться курсор во время выполнения транзакций. Если это значение равно true, курсоры не закрываются при фиксации и откате транзакций. Все ресурсы, полученные в течение единицы работы, блокируются, но блокировки строк и объектов, неявно установленные во время выполнения единицы работы, снимаются.	нет	″true″ ″false″	″true″
″cursor sensitivity″	Задает чувствительность курсора при отправке запроса в базу данных. Работа этого свойства зависит от resultSetType: • ResultSet.TYPE_FORWARD_ONLY или ResultSet.TYPE_SCROLL_SENSITIVE означает, что значение этого свойства управляет чувствительностью курсора, запрашиваемой программой на Java у базы данных. • ResultSet.TYPE_SCROLL_INSENSITIVE указывает, что это свойство должно игнорироваться. Данное свойство игнорируется при подключении к i5/OS V5R1 и более ранних версий.	нет	″asensitive″ ″sensitive″ ″insensitive″	″asensitive″
″database name″	Задает базу данных, с которой будет установлено соединение; поддерживаются базы данных в независимом пуле вспомогательной памяти. Это свойство применимо только при подключении к i5/OS версии V5R2 или более поздней. Если указано имя базы данных, это имя должно существовать в каталоге реляционных баз данных сервера. Доступ к базе данных определяется следующими критериями: • Если в данном свойстве указано имя базы данных, используется указанная база данных. Если такая база данных не существует, соединение не устанавливается. • Если в данном свойстве указано *SYSBAS, применяется системная база данных по умолчанию. • Если данное свойство не задано, имя базы данных будет получено из описания задания, указанного в пользовательском профайле. Если в описании задания не указано имя базы данных, применяется системная база данных по умолчанию.	нет	Имя базы данных *SYSBAS	Имя базы данных, указанное в описании задания для польз. профайла. Если в описании задания не указано имя базы данных, применяется системная база данных по умолчанию.

Property	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″decfloat rounding mode″	Specifies the rounding mode to use when working with the decfloat data type. This property is ignored when connecting to systems running V5R4 and earlier.	нет	″half even″ ″half up″ ″down″ ″ceiling″ ″floor″ ″half down″ ″up″	″half even″

Property	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″libraries″	Задает одну или несколько библиотек, добавляемых в список библиотек задания сервера, а также библиотеку (схему) по умолчанию. **Список библиотек** С помощью указанных библиотек сервер преобразует простые имена сохраненных процедур; имена преобразуются процедурами. При перечислении библиотеки следует разделять запятыми или пробелами. You can use *LIBL as a placeholder for the current library list of the server job • Если в качестве первой записи указано *LIBL, то перечисленные библиотеки добавляются в текущий список библиотек задания сервера. • Если запись *LIBL не указана, то перечисленные библиотеки заменяют собой текущий список библиотек задания сервера. Дополнительные сведения см. в разделе Свойство LibraryList JDBC. **Схема по умолчанию** Простые имена в операторах SQL преобразуются на сервере с помощью схемы по умолчанию. Например, оператор ″SELECT * FROM MYTABLE″ предполагает, что таблица MYTABLE находится в схеме по умолчанию. Схему по умолчанию можно задать в URL соединения. Если это не сделано, то в зависимости от применяемого соглашения поиск выполняется следующим образом: • **Соглашение о присвоении имен SQL** Если в URL соединения не задана схема по умолчанию, то: – Схемой по умолчанию становится первая запись (если она отлична от *LIBL) – Если в первой записи указано *LIBL, то схемой по умолчанию становится вторая запись – Если это свойство не задано или указано только *LIBL, то по умолчанию применяется профайл пользователя • **Системное соглашение о присвоении имен** Если в URL соединения не задана схема по умолчанию, то: – Схема по умолчанию не задается, а при поиске объектов с неполными именами применяются перечисленные библиотеки – Если это свойство не задано или указано только *LIBL, то при поиске объектов сервер применяет текущий список библиотек задания	нет	Список библиотек сервера (разделители - запятые или пробелы)	″*LIBL″

Property	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″maximum precision″	Задает максимальную точность операций с десятичными числами, которая может применяться базой данных.	нет	″31″ ″63″	″31″
″maximum scale″	Задает максимальную длину дробной части числа, которая может применяться в базе данных.	нет	″0″-″63″	″31″
″minimum divide scale″	Задает минимальную длину дробной части при делении чисел.	нет	″0″ ″1″ ″2″ ″3″ ″4″ ″5″ ″6″ ″7″ ″8″ ″9″	″0″
″ccsid пакета″	Задает кодировку символов для пакета SQL и операторов, выполняемых на сервере.	нет	″1200″ (UCS-2) ″13488″ (UTF-16)	″13488″
″transaction isolation″	Задает уровень независимости транзакции по умолчанию.	нет	″none″ ″read uncommited″ ″read commited″ ″repeatable read″″serializable″	″read uncommitted″
″translate hex″	Задает способ интерпретации шестнадцатеричных литералов.	нет	″character″ (Interpret hexadecimal literals as character data) ″binary″ (Interpret hexadecimal literals as binary data)	″character″
″true autocommit″	Указывает, что соединение должно использовать истинную автоматическую фиксацию. Это значит, что включена автоматическая фиксация и применяется уровень изоляции, отличный от *NONE. По умолчанию драйвер использует автоматическую фиксацию с применением на сервере уровня изоляции *NONE.	нет	″true″ (Use true autocommit.) ″false″ (Do not use true autocommit.)	″false″

Property	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″xa loosely coupled support″	Указывает, разрешено ли применять общие блокировки для ветвей транзакций со слабой связью. **Примечание:** Этот параметр игнорируется в i5/OS V5R3 или более ранних.	нет	″0″ = запретить применение общих блокировок ″1″ = разрешить применение общих блокировок	″0″

Свойства формата

Свойства формата задают формат и разделители даты и времени, а также соглашения о присвоении имен, применяемые с операторами SQL.

Property	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″date format″	Задает формат даты, применяемый в операторах SQL.	нет	″mdy″ ″dmy″ ″ymd″ ″usa″ ″iso″ ″eur″ ″jis″ ″julian″	(задание сервера)
″date separator″	Задает разделитель дат, применяемый в литералах операторов SQL. Для применения этого свойства date format необходимо установить в значение julian, mdy, dmy или ymd.	нет	″/″ (косая черта) ″-″ (тире) ″.″ (точка) ″,″ (запятая) ″b″ (пробел)	(задание сервера)
″decimal separator″	Задает десятичный разделитель для числовых литералов в операторах SQL.	нет	″.″ (точка) ″,″ (запятая)	(задание сервера)
″naming″	Задает правила присвоения имен таблицам.	нет	″sql″ (формат `схема.таблица`) ″system″ (формат `схема/таблица`)	″sql″
″time format″	Задает формат литералов времени в операторах SQL.	нет	″hms″ ″usa″ ″iso″ ″eur″ ″jis″	(задание сервера)
″time separator″	Задает разделитель для литералов времени в операторах SQL. Для применения этого свойства необходимо присвоить свойству time format значение hms.	нет	″:″ (двоеточие) ″.″ (точка) ″,″ (запятая) ″b″ (пробел)	(задание сервера)

Свойства производительности

Свойства производительности - это атрибуты кэширования, преобразования и сжатия данных, предварительной выборки и т.п.

Property	Описание	Обяза- тельное	Допустимые значения	Значение по умолчанию
″big decimal″	Указывает, будет ли применяться промежуточный объект java.math.BigDecimal для преобразования в упакованный десятичный и зонный десятичный форматы. Если это свойство равно true, при преобразовании в зонный десятичный и упакованный десятичный форматы создается промежуточный объект java.math.BigDecimal, как описано в документации по JDBC. Если это свойство равно false, при преобразовании в зонный десятичный и упакованный десятичный форматы промежуточные объекты не применяются. Вместо этого такие значения напрямую преобразуются в значения Java типа double и обратно. Такое преобразование будет выполняться быстрее, но может не соответствовать всем правилам преобразования и усечения данных, указанным в спецификации JDBC.	нет	″true″ ″false″	″true″
″block criteria″	Указывает критерий для получения данных от сервера в виде блоков записей. Ненулевое значение данного свойства уменьшит число обращений к серверу и повысит его производительность. Убедитесь, что объединение записей в блоки отключено, если курсор впоследствии будет использован для выполнения операций обновления. В противном случае обновляемая строка может не совпадать с текущей.	нет	″0″ (не записывать блоками) ″1″ (записывать блоками, только если указано значение FOR FETCH ONLY) ″2″ (записывать блоками, только если не указано значение FOR UPDATE)	″2″
″block size″	Задает размер блока записей (в килобайтах), который будет считываться с сервера и кэшироваться на клиенте. Это свойство игнорируется, если свойство block criteria равно нулю. Увеличение размера блоков приводит к уменьшению числа обращений к серверу и, таким образом, способствует повышению производительности.	нет	″0″ ″8″ ″16″ ″32″ ″64″ ″128″ ″256″ ″512″	″32″

Property	Описание	Обяза- тельное	Допустимые значения	Значение по умолчанию
″data compression″	Указывает, сжимаются ли данные набора результатов. Если это свойство равно true, конечный набор данных сжимается. Если это свойство равно false, конечный набор данных не сжимается. Сжатие данных может повысить производительность при получении большого объема данных.	нет	″true″ ″false″	″true″
″extended dynamic″	Указывает, будет ли применяться расширенная динамическая поддержка. Расширенная динамическая поддержка позволяет заносить в кэш сервера операторы динамического SQL. При первой подготовке конкретного оператора SQL он сохраняется в пакете SQL на сервере. Если пакет не существует, он автоматически создается. При последующих подготовках того же оператора SQL сервер применяет информацию из пакета SQL, что позволяет значительно сократить время обработки оператора. Если это свойство равно true, то необходимо с помощью свойства property указать имя пакета.	нет	″true″ ″false″	″false″
″lazy close″	Указывает, будет ли откладываться закрытие курсора до получения следующего запроса. Это позволит повысить производительность за счет уменьшения числа запросов к серверу.	нет	″true″ ″false″	″false″
″lob threshold″	Задает максимальный размер объекта LOB (в байтах), который может быть получен в составе набора результатов. Если размер LOB больше указанного значения, то он возвращается по частям, что увеличивает длительность обмена данными с сервером. Чем больше пороговый размер LOB, тем меньше число обращений к серверу, но тем больше объем загружаемых данных, часть из которых может оказаться ненужной. При низком пороговом размере LOB число обращений к серверу будет довольно велико, однако вы будете получать только те данные LOB, которые действительно нужны.	нет	″0″ - ″16777216″	″32768″
″package″	Указывает основное имя пакета SQL. Обратите внимание, что при создании имени пакета SQL на сервере используются только первые семь символов. Это свойство применяется только в том случае, если свойству extended dynamic присвоено значение true. Кроме того, значение этого свойства обязательно должно быть задано, если свойство extended dynamic равно true.	нет	Пакет SQL	″″

Property	Описание	Обяза- тельное	Допустимые значения	Значение по умолчанию
″package add″	Указывает, следует ли добавлять новые подготовленные операторы в пакет SQL, указанный в свойстве package. Это свойство применяется только в том случае, если свойству extended dynamic присвоено значение true.	нет	″true″ ″false″	″true″
″package cache″	Это свойство указывает, следует ли кэшировать подмножество данных пакета SQL в памяти клиентов. Кэширование пакетов SQL на локальном компьютере иногда позволяет уменьшить число обращений к серверу для подготовки и описания операторов. Это свойство применяется только в том случае, если свойству extended dynamic присвоено значение true.	нет	″true″ ″false″	″false″
″package criteria″	Задает тип операторов SQL, для которых предназначен этот пакет SQL. Это свойство позволяет ускорить обработку составных условий соединения. Это свойство применяется только в том случае, если свойству extended dynamic присвоено значение true.	нет	default (в пакете хранятся только операторы SQL с маркерами параметров) ″select″ (в пакете хранятся все операторы SQL SELECT)	″default″
″package error″	Задает действие, выполняемое в ответ на возникновение ошибки при работе с пакетами SQL. При возникновении ошибки пакета SQL драйвер, в зависимости от значения этого свойства, может создать дополнительный объект SQLException или отправить уведомление объекту Connection. Это свойство применяется только в том случае, если свойству extended dynamic присвоено значение true.	нет	″exception″ ″warning″ ″none″	″warning″
″package library″	Задает библиотеку пакета SQL. Это свойство применяется только в том случае, если свойству extended dynamic присвоено значение true.	нет	Библиотека пакета SQL	″QGPL″
″prefetch″	Указывает, нужно ли выполнять предварительную выборку перед выполнением оператора SELECT. Это позволит быстрее получить первые строки таблицы ResultSet.	нет	″true″ ″false″	″true″

	Property	Описание	Обяза- тельное	Допустимые значения	Значение по умолчанию
R R R R R R R	″qaqqinilib″	Задает имя библиотеки QAQQINI. С его помощью можно задать библиотеку, в которой расположен нужный файл qaqqini. A qaqqini file contains all of the attributes that can potentially impact the performance of the DB2 for i5/OS database engine.	нет	″QAQQINI library name″	(значение сервера по умолчанию)
	″query optimize goal″	Задает цель, которую должен применять сервер для оптимизации запросов. Этот параметр соответствует опции QAQQINI сервера OPTIMIZATION_GOAL. **Примечание:** Данное свойство игнорируется при подключении к i5/OS V5R3 и более ранних версий.	нет	″0″ = Оптимизировать запрос для первого блока данных (*FIRSTIO), когда используются расширенные динамические пакеты; Оптимизировать запрос для всего набора результатов (*ALLIO), когда пакеты не используются ″1″ = Оптимизировать запрос для первого блока данных (*FIRSTIO) ″2″ = Оптимизировать запрос для всего набора результатов (*ALLIO)	″0″
+ + + + + +	″query storage limit″	Limits the storage used by a query. This property compares the storage limit you specify to the estimated storage usage of the query. If the estimated storage usage exceeds the specified storage limit, the query is not allowed to execute.	нет	-1 (*NOMAX) to 2 147 352 578	-1, which indicates a special value of *NOMAX
+ + + + + +	″receive buffer size″	Specifies the buffer size used to receive data through the socket connection between the front-end driver and the system. **Примечание:** This does not specify the actual receive buffer size. It is only used as a hint by the underlying socket code.	нет	″1″ - max size	(platform dependent)

Property	Описание	Обяза- тельное	Допустимые значения	Значение по умолчанию
″send buffer size″	Specifies the buffer size used to send data through the socket connection between the front-end driver and the system. **Примечание:** This does not specify the actual send buffer size. It is only used as a hint by the underlying socket code.	нет	″1″ - max size	(platform dependent)
″variable field compression″	Specifies whether variable-length fields should be compressed.	нет	″true″ ″false″	″true″

Свойства сортировки

Свойства сортировки управляют сохранением и сортировками.

Свойство сортировки	Описание	Обяза- тельное	Допустимые значения	Значение по умолчанию
″sort″	Указывает, каким образом сервер должен отсортировать записи перед их отправкой клиенту.	нет	″hex″ (отсортировать по 16-ричным значениям) ″language″ (отсортировать по набору языков в свойстве ″sort language″) ″table″ (отсортировать на основе параметров таблицы сортировки свойства ″sort table″)	″hex″
″sort language″	Задает трехсимвольный ИД языка для выбора последовательности сортировки. Это свойство игнорируется, если свойству ″sort″ не присвоено значение ″language″.	нет	ИД языка	ENU
″sort table″	Задает библиотеку и файл сервера, в котором хранится таблица сортировки. Это свойство игнорируется, если свойству ″sort″ не присвоено значение ″table″.	нет	Полное имя таблицы сортировки	″″
″sort weight″	Указывает, должен ли сервер учитывать регистр символов при сортировке записей. This property has no effect unless the ″sort″ property is set to ″language.″	нет	″shared″ (upper- and lowercase characters sort as the same character) ″unique″ (upper- and lowercase characters sort as different characters)	″shared″

Прочие свойства

Прочие свойства, для которых не была выделена отдельная категория. Эти свойства задают драйвер JDBC, опции уровня доступа к базе данных, тип двунаправленных строк, параметры усечения данных и т.п.

Свойство	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″access″	Задает уровень доступа к базе данных, предоставленный соединению.	нет	″all″ (все допустимые операторы SQL) ″read call″ (только операторы SELECT CALL) ″read only″ (только операторы SELECT)	″all″
‡ ″autocommit exception″ ‡	Specifies whether to throw an SQLException when Connection.commit() is called if autocommit is enabled.	нет	″true″ ″false″	″false″
″bidi string type″	Задает тип строки вывода двунаправленных данных. Дополнительная информация приведена в разделе BidiStringType.	нет	″″ (использовать CCSID для определения типа двунаправленной строки) ″0″ (тип строки по умолчанию для недвунаправленной строки (LTR)) ″4″ ″5″ ″6″ ″7″ ″8″ ″9″ ″10″ ″11″	″″
″bidi implicit reordering″	Указывает, должно ли применяться неявное переупорядочение LTR-RTL.	нет	″true″ ″false″	″true″
″bidi numeric ordering″	Указывает, должна ли применяться функция кольцевого упорядочения чисел.	нет	″true″ ″false″	″false″

Свойство	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″data truncation″	Указывает, будут ли при усечении **символьных данных** создаваться уведомления и исключения. Если это свойство равно true, действуют следующие ограничения: • Запись в базу данных усеченных символьных данных приводит к возникновению исключительной ситуации • Указание усеченных данных в запросе приводит к выдаче предупреждения. Если это свойство равно false, запись усеченных данных в базу данных или указание их в запросах не создает исключительной ситуации или предупреждения. По умолчанию используется значение true. Данное свойство не влияет на **числовые данные**. Запись в базу данных усеченных числовых данных всегда приводит к ошибке; указание усеченных данных в запросе всегда приводит к выдаче предупреждения.	нет	″true″ ″false″	″true″
R ″driver″	Задает реализацию драйвера JDBC. Драйвер JDBC IBM Toolbox for Java в зависимости от среды может использовать различные реализации драйвера JDBC. If the environment is an i5/OS JVM on the same server as the database to which the program is connecting, the native IBM Developer Kit for Java JDBC driver can be used. In any other environment, the IBM Toolbox for Java JDBC driver is used. Это свойство игнорируется, если не задано свойство ″secondary URL″.	нет	″toolbox″ (use only the IBM Toolbox for Java JDBC driver) ″native″ (use the IBM Developer Kit for Java JDBC driver if running on the server, otherwise use the IBM Toolbox for Java JDBC driver)	″toolbox″
″errors″	Задает степень подробности сообщений об ошибках сервера.	нет	″basic″ ″full″	″basic″

Свойство	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″extended metadata″	Указывает, должен ли драйвер запрашивать с сервера расширенные метаданные. Указание для этого свойства значения true увеличивает точность информации, возвращаемой следующими методами ResultSetMetaData: • getColumnLabel(int) • isReadOnly(int) • isSearchable(int) • isWriteable(int) Кроме того, при установке этого свойства включается поддержка метода ResultSetMetaData.getSchemaName(int). Однако, включение этого свойства снижает производительность, поскольку требует от сервера получения большего объема информации. Если получение от перечисленных методов дополнительной информации не требуется, оставьте для данного свойства значение по умолчанию (false). В частности, если это свойство равно false, метод ResultSetMetaData.isSearchable(int) всегда возвращает значение ″true″, поскольку у драйвера недостаточно информации для принятия решения. Включение данного свойства вынуждает драйвер получить от сервера необходимые данные. Расширенные метаданные поддерживаются только при подключении к серверу под управлением i5/OS версии V5R2 или более поздней.	нет	″true″ ″false″	″false″
″full open″	Указывает, будет ли сервер полностью открывать файл для каждого запроса. По умолчанию сервер оптимизирует запросы на открытие файлов. Эта оптимизация повышает производительность, но может вызвать сбой при повторной обработке запроса, если в системе запущен монитор базы данных. Значение true может применяться во время работы монитора только в том случае, если все запросы полностью идентичны.	нет	″true″ ″false″	″false″
″hold input locators″	Указывает как должны выделяться локаторы ввода как блокированные или нет. Если выбраны локаторы блокированного типа, то после выполнения фиксации они не будут освобождаться.	нет	″true″ (блокированный тип) ″false″	″true″
″hold statements″	Указывает, должны ли операторы оставаться открытыми до границы транзакции если автоматическая фиксация выключена и операторы связаны с локатором LOB. По умолчанию при закрытии оператора все связанные с ним ресурсы освобождаются. Значение true следует присваивать этому свойству лишь в том случае, если локатор LOB необходим после закрытия оператора.	нет	″true″ ″false″	″false″

Свойство	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″keep alive″	Specifies whether socket connection is to be periodically checked for operational status.	нет	″true″ ″false″	(platform dependent)
″key ring name″	Задает имя класса набора ключей, который должен применяться для установления соединения SSL с сервером. Это свойство игнорируется, если свойство secure не равно true, либо в свойстве key ring password не задан пароль к связке ключей.	нет	″key ring name″	″″
″key ring password″	Задает пароль для класса набора ключей, применяемого для установления соединения SSL с сервером. Это свойство игнорируется, если свойство secure не равно true, либо в свойстве key ring name не задано имя связки ключей.	нет	″key ring password″	″″
″metadata source″	Specifies how to retrieve DatabaseMetaData. If set to ″0,″ database metadata will be retrieved through the Retrieve Object Information (ROI) data flow. If set to ″1,″ database metadata will be retrieved by calling system stored procedures. The only method that is available using stored procedures is getColumnPrivileges().	нет	″0″ (ROI access) ″1″ (SQL stored procedures)	″1″
″proxy server″	Задает имя хоста и порт системы среднего уровня, в которой работает сервер Proxy. Значение свойства задается в формате *hostname*[:*port*], где порт - необязательное значение. Если значение свойства не задано, применяются имя хоста и порт из свойства *com.ibm.as400.access.AS400.proxyServer*. Номер порта по умолчанию равен *3470* (для соединений SSL номер порта по умолчанию - *3471*). В системе среднего уровня должен работать сервер Proxy. В двухуровневой среде имя системы среднего уровня игнорируется.	нет	имя хоста и порт сервера Proxy	(значение свойства proxyServer или ″none″, если оно не задано)
″remarks″	Задает источник текста, который должен подставляться в столбец REMARKS таблицы ResultSets, возвращаемой методом DatabaseMetaData.	нет	″sql″ (комментарий к объекту SQL) ″system″ (описание объекта i5/OS)	″system″
″secondary URL″	Задает URL для установления соединения с помощью драйвера DriverManager среднего уровня в многоуровневой среде. This property allows you to use this driver to connect to other databases. В качестве escape-символа перед обратной косой чертой и точкой с запятой в URL следует указывать обратную косую черту.	нет	URL JDBC	(текущий URL JDBC)
″secure″	Указывает, применяется ли для подключения к серверу протокол SSL. Соединение SSL может быть установлено только с серверами версии V4R4 и выше.	нет	″true″ (использовать шифрование для всех соединений клиент-сервер) ″false″ (шифровать только пароли)	″false″

Свойство	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″server trace″	Задает уровень трассировки задания сервера JDBC. Если трассировка включена, ее начало совпадает с подключением клиента к серверу, а прекращение происходит в момент разрыва соединения. Для трассировки соединения трассировка должна быть включена перед установлением соединения.	нет	″0″ (трассировка не активирована) ″2″ (запустить монитор базы данных для задания сервера JDBC) ″4″ (запустить отладку задания сервера JDBC) ″8″ (сохранить протокол задания после завершения задания сервера JDBC) ″16″ (запустить трассировку задания для задания сервера JDBC) ″32″ (сохранить информацию SQL) ″64″ (поддерживает активацию трассировки на хосте базы данных) Можно указать одновременно несколько флагов сортировки, сложив указанные значения. Например, если указать значение ″6″, будут запущены монитор базы данных и отладка.	″0″
″thread used″	Указывает, будут ли применяться нити при работе с серверами хоста.	нет	″true″ ″false″	″true″

Свойство	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″toolbox trace″	Specifies what category of an IBM Toolbox for Java trace to log. Сообщения трассировки служат для отладки программ, использующих вызовы JDBC. Однако занесение в протокол сообщений трассировки отрицательно сказывается на производительности, поэтому значение свойства true следует задавать только для отладки. Сообщения трассировки заносятся в поток вывода System.out.	нет	″″ ″none″ ″datastream″ (заносить в протокол сведения о передаче данных между локальным хостом и удаленной системой) ″diagnostic″ (заносить в протокол сведения о состоянии объектов) ″error″ (заносить в протокол ошибки, которые приводят к исключительным ситуациям) ″information″ (служит для отслеживания средств управления в исполняемом коде) ″warning″ (заносить в протокол исправимые ошибки) ″conversion″ (заносить в протокол преобразования наборов символов из Unicode в локальные кодировки и обратно) ″proxy″ (заносить в протокол сведения о передаче данных между клиентом и сервером proxy) ″pcml″ (задает способ преобразования PCML данных, передаваемых и получаемых с сервера) ″jdbc″ (заносить в протокол сведения jdbc) ″all″ (заносить в протокол все категории данных) ″thread″ (заносить в протокол информацию нитей)	″″

Свойство	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″trace″	Указывает, будут ли заноситься в протокол сообщения трассировки. Сообщения трассировки служат для отладки программ, использующих вызовы JDBC. Однако занесение в протокол сообщений трассировки отрицательно сказывается на производительности, поэтому значение свойства true следует задавать только для отладки. Сообщения трассировки заносятся в поток вывода System.out.	нет	″true″ ″false″	″false″
″translate binary″	Указывает, преобразуются ли двоичные данные. Если это свойство равно true, поля BINARY и VARBINARY обрабатываются как CHAR и VARCHAR.	нет	″true″ ″false″	″false″
″translate boolean″	Specifies how Boolean objects are interpreted when setting the value for a character field/parameter using the PreparedStatement.setObject(), CallableStatement.setObject() or ResultSet.updateObject() methods. Setting the property to ″true″ would store the Boolean object in the character field as either ″true″ or ″false.″ Setting the property to ″false″ would store the Boolean object in the character field as either ″1″ or ″0.″	нет	″true″ ″false″	″true″

Ссылки, связанные с данной

“Регистрация драйвера JDBC” на стр. 79
Before using JDBC to access data in a server database file, you need to register the JDBC driver for the IBM Toolbox for Java licensed program with the DriverManager.

Свойство Librarylist JDBC

Свойство LibraryList JDBC позволяет задавать одну или несколько библиотек, которые следует добавить или заменить в списке библиотек задания севера, а также дополнительно выбрать библиотеку по умолчанию (схема по умолчанию).

В примерах в таблице ниже предполагается, что:

- Библиотека MYLIBDAW содержит объект MYFILE_DAW
- Вам необходимо выполнить следующий оператор SQL:

```
"SELECT * FROM MYFILE_DAW"
```

Сценарий	Соглашение о присвоении имен SQL	Системное соглашение о присвоении имен
Основные правила	Поиск выполняется только в одной библиотеке. • Если URL содержит имя библиотеки, то будет применяться эта библиотека. Она используется в качестве библиотеки по умолчанию. • Если URL не содержит имен библиотек, то будет применяться первая библиотека из свойства 'библиотеки'. Она используется в качестве библиотеки по умолчанию. • Если в URL и свойстве библиотек не указано имен библиотек, будет применяться библиотека, имя которой совпадает с именем текущего пользовательского профайла. В список библиотек задания добавляются библиотеки, перечисленные в свойстве библиотек. От этого может зависеть работа некоторых триггеров и сохраненных процедур. Использование простых имен в операторах от этого не зависит.	В список библиотек задания добавляются библиотеки, перечисленные в свойстве библиотек. Если библиотека по умолчанию указана в URL, то эта библиотека будет использована по умолчанию.
1. Имена библиотек не указаны ни в одном из объектов.	В схеме по умолчанию используется библиотека пользовательского профайла.	Если схема по умолчанию не определена, выполняется поиск в списке библиотек задания.
2. Библиотека по умолчанию указана в URL.	В схеме по умолчанию используется указанная библиотека.	В схеме по умолчанию используется указанная библиотека.Поиск в списке библиотек для преобразования простых имен в операторах SQL не выполняется.
3. Библиотека по умолчанию задана в свойстве библиотек.	В схеме по умолчанию используется указанная библиотека.	Если схема по умолчанию не определена, выполняется поиск всех библиотек в списке.
4. Библиотека по умолчанию указана в URL и свойстве библиотек.	В схеме по умолчанию используется библиотека из URL. Список библиотек игнорируется.	В схеме по умолчанию используется библиотека из URL. Поиск в списке библиотек для преобразования простых имен в операторах SQL не выполняется.
5. В свойстве библиотек задана библиотека по умолчанию с неправильным именем	В схеме по умолчанию используется указанная библиотека	Если схема по умолчанию не определена, использовать список библиотек невозможно, поскольку одно из имен указано неверно. В этом случае применяется список библиотек задания.
6. В URL нет имени библиотеки; библиотека задана в свойстве библиотек и файл расположен во второй библиотеке списка	В схеме по умолчанию используется первая библиотека в списке, а остальные библиотеки игнорируются.	В случае, если библиотеки есть во всех объектах, то схема по умолчанию не применяется, выполняется поиск всех библиотек в списке и этот список заменяет список библиотек задания. Если одной из библиотек в списке нет, список библиотек задания не изменяется.
7. В свойстве библиотек указаны библиотеки, но список начинается с запятой	В схеме по умолчанию применяется пользовательский профайл	Если схема по умолчанию не определена, выполняется поиск всех библиотек в списке и этот список заменяет список библиотек задания.
8. В свойстве библиотек указаны библиотеки и список начинается со значения *LIBL	В схеме по умолчанию применяется пользовательский профайл	Если схема по умолчанию не определена, выполняется поиск всех библиотек в списке и указанные библиотеки добавляются в конец списка

Сценарий	Соглашение о присвоении имен SQL	Системное соглашение о присвоении имен
9. В свойстве библиотек указаны библиотеки и список заканчивается значением *LIBL	В схеме по умолчанию применяется первая библиотека списка, все остальные библиотеки игнорируются	Если схема по умолчанию не определена, выполняется поиск всех библиотек в списке и указанные библиотеки добавляются в начало списка библиотек задания
10. В URL указано неправильное имя библиотеки	Если схема по умолчанию не определена, применяется библиотека, соответствующая пользовательскому профайлу	Если схема по умолчанию не определена, применяется список библиотек пользовательского профайла

Примечание: Если в URL указана схема по умолчанию и свойство библиотек не применяется, схема по умолчанию добавляется в начало текущего списка библиотек

Типы SQL JDBC

Не все типы SQL, описанные в спецификации JDBC, поддерживаются DB2 для i5/OS.

Неподдерживаемые типы SQL

Неподдерживаемые типы SQL драйвер JDBC заменяет на аналогичные.

В следующей таблице перечислены неподдерживаемые типы SQL и типы, на которые они заменяются драйвером JDBC.

Неподдерживаемый тип SQL	Применяемый тип SQL
BIT	SMALLINT
TINYINT	SMALLINT
LONGVARCHAR	VARCHAR
LONGVARBINARY	VARBINARY

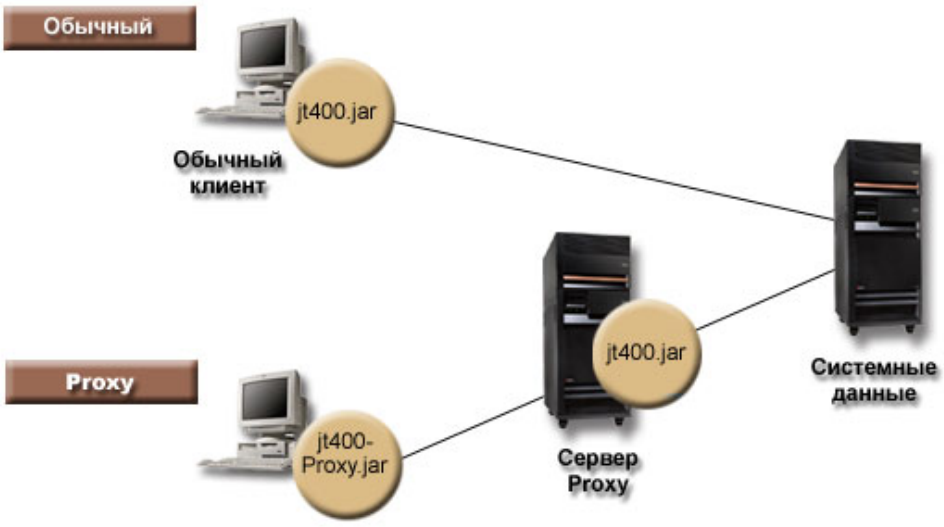
Поддержка Proxy

Некоторые классы IBM Toolbox for Java обеспечивают поддержку Proxy. Сервер Proxy позволяет выполнять задачи IBM Toolbox for Java на виртуальной машине Java (JVM), отличной от той, где запускается приложение.

Классы Proxy хранятся в файле jt400Proxy.jar и поставляются в составе IBM Toolbox for Java. The proxy classes, like the other classes in the IBM Toolbox for Java, comprise a set of platform independent Java classes that can run on any computer with a Java virtual machine. Классы Proxy отправляют все вызовы методов приложению сервера или серверу Proxy. На сервере Proxy реализован полный набор всех классов IBM Toolbox for Java. Если клиент использует класс Proxy, то запрос передается на сервер Proxy, который создает реальные объекты IBM Toolbox for Java и управляет ими.

На рис. 1 показана схема соединения обычного клиента и клиента Proxy с сервером. The proxy server can be the System i that contains the data.

Рис. 1: Схема соединения обычного клиента и клиента Proxy с сервером



Приложение, применяющее поддержку Proxy, выполняется медленнее, чем при работе с обычными классами IBM Toolbox for Java, так как для поддержки небольших классов Proxy требуются дополнительные соединения. Чем меньше методов вызывает приложение, тем меньше будет для него ощущаться снижение производительности системы.

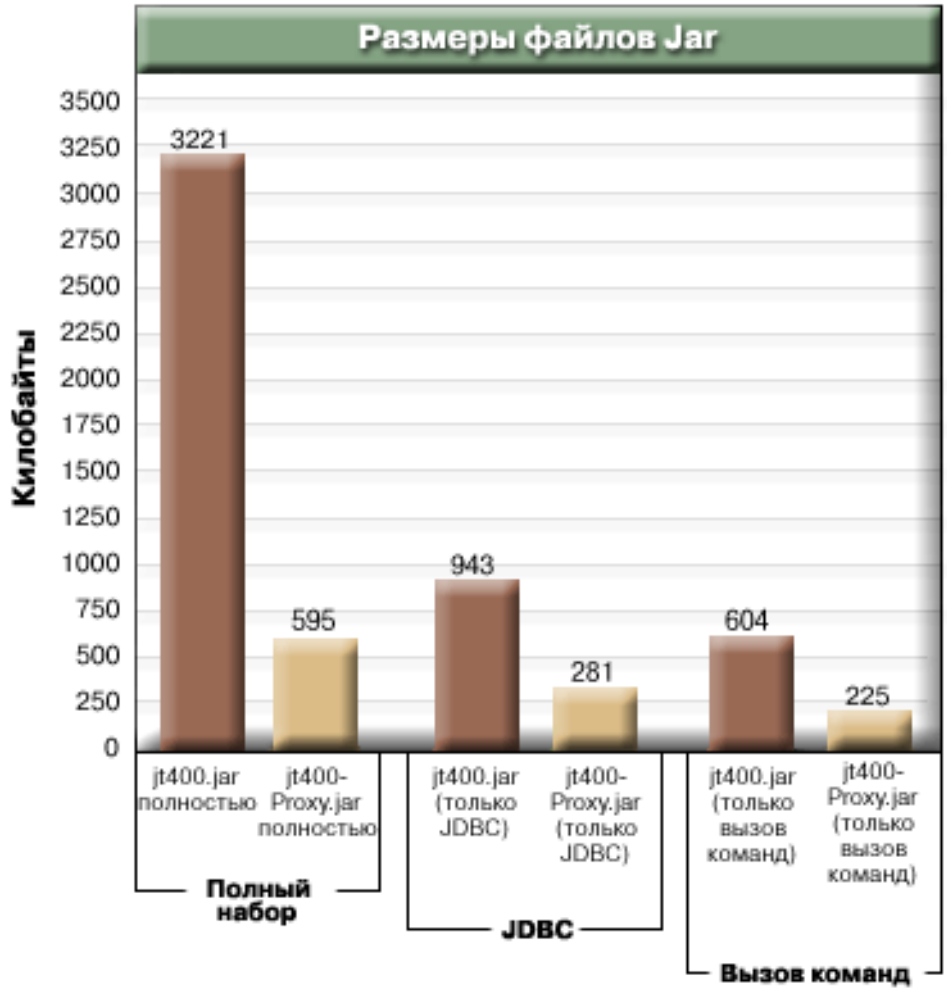
До появления поддержки Proxy классы, содержащие общий интерфейс, все классы обработки запросов и само приложение запускались на одной виртуальной машине Java (JVM). При использовании поддержки Proxy, классы общего интерфейса должны работать на одной виртуальной машине с приложением, а классы обработки запросов могут работать на отдельной виртуальной машине Java. Поддержка Proxy не изменяет общий интерфейс. Одна и та же программа может запускаться как в версии с поддержкой Proxy, так и в стандартной версии IBM Toolbox for Java.

Применение файла jt400Proxy.jar

Цель многоуровневого сценария Proxy состоит в создании файла .jar минимально возможного размера, так чтобы загрузка этого файла с общим интерфейсом из аплета занимала как можно меньше времени. При использовании классов Proxy не требуется устанавливать на клиенте весь пакет IBM Toolbox for Java полностью. Вместо этого компоновщик AS400JarMaker позволяет включить в состав файла jt400Proxy.jar только необходимые компоненты, сокращая размер этого файла до минимума.

Приведенный ниже рис. 2 позволяет сравнить размеры Proxy-файлов .jar и стандартных файлов .jar:

Рис. 2: Сравнение размеров Proxy-файлов .jar и стандартных файлов .jar



Дополнительное преимущество применения поддержки Proxy заключается в снижении числа открытых портов в брандмауэре. Для использования стандартных классов IBM Toolbox for Java необходимо открыть несколько портов. Это связано с тем, что каждая служба IBM Toolbox for Java использует отдельный порт для обмена данными с сервером. Например, службы вызова команды, сетевой печати, JDBC и т.д. используют разные порты. Для передачи данных через каждый из этих портов необходимо применять брандмауэр. Если включена поддержка Proxy, все данные передаются через один и тот же порт.

Стандартный Proxy и туннели HTTP

Существует два варианта работы с поддержкой Proxy - стандартный Proxy и туннели HTTP:

- При стандартном Proxy клиент и сервер Proxy обмениваются данными с помощью сокета через определенный порт. По умолчанию это порт 3470. Для изменения номера порта можно использовать метод setPort() или указать при запуске сервера Proxy опцию `-port`.Например:

  ```
  java com.ibm.as400.access.ProxyServer -port 1234
  ```

- При использовании туннелей HTTP обмен данными между клиентом и сервером Proxy осуществляется посредством сервера HTTP. IBM Toolbox for Java содержит сервлет, предназначенный для обработки запросов Proxy. Клиент Proxy вызывает сервлет с помощью сервера HTTP. Преимущество туннелей заключается в том, что нет необходимости открывать дополнительный порт через брандмауэр, так как обмен данными происходит через порт HTTP. Недостатком же этого метода является меньшая скорость работы.

В IBM Toolbox for Java применяемый метод определяется именем сервера Proxy:

- При стандартном Proxy используется имя сервера. Например:

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

- При использовании туннелей применяется URL. Например:

  ```
  com.ibm.as400.access.AS400.proxyServer=http://myServer
  ```

При использовании стандартного Proxy между клиентом и сервером устанавливается соединение с помощью сокетов. При сбое соединения сервер освобождает ресурсы, связанные с клиентом.

При использовании протокола HTTP соединение Proxy не устанавливается. Для каждого потока данных создается новое соединение. При этом у сервера нет информации о том, завершена ли работа клиента. Таким образом, сервер не знает, когда следует освободить ресурсы. Сервер туннелей решает эту проблему с помощью отдельной нити, освобождающей ресурсы через заранее определенный интервал времени (основанный на значении тайм-аута).

По истечении заданного интервала времени запускается нить, которая освобождает давно не использовавшиеся ресурсы. Работа нити определяется двумя системными свойствами:

- com.ibm.as400.access.TunnelProxyServer. clientCleanupInterval задает интервал времени в секундах, с которым запускается нить очистки ресурсов. Значение по умолчанию - каждые два часа.
- com.ibm.as400.access.TunnelProxyServer. clientLifetime определяет в секундах, как долго ресурс может не использоваться перед тем, как он будет освобожден. Значение по умолчанию - 30 минут.

Работа с сервером Proxy

Для того чтобы использовать реализацию классов IBM Toolbox for Java для сервера Proxy, выполните следующие действия:

1. Запустите AS400ToolboxJarMaker с файлом jt400Proxy.jar, чтобы отключить ненужные вам классы. Это действие необязательное, но рекомендуется его выполнить.
2. Передайте jt400Proxy.jar клиенту. Если используется аплет Java, то можно загружать файл .jar с сервера HTML.
3. Решите, какой сервер станет сервером Proxy.
   - В случае приложений на Java этим сервером может быть любой компьютер.
   - В случае аплетов Java сервер Proxy должен запускаться на том же компьютере, что и сервер HTTP.
4. Убедитесь, что путь к файлу jt400.jar указан в переменной CLASSPATH на сервере.
5. Запустите сервер Proxy или воспользуйтесь сервлетом Proxy:
   - Для использования стандартного Proxy запустите сервер Proxy с помощью следующей команды:

     ```
     java com.ibm.as400.access.ProxyServer
     ```

   - Для использования Proxy с туннелями настройте сервер HTTP для работы с сервлетом Proxy. Имя класса сервлета - com.ibm.as400.access.TunnelProxyServer, он находится в файле jt400.jar.
6. На клиенте настройте системное свойство, обозначающее сервер Proxy. В IBM Toolbox for Java применяемый метод определяется этим свойством:
   - При использовании стандартного Proxy значение свойства -это имя системы, в которой работает сервер Proxy. Например:

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

   - При использовании туннелей применяется URL. Например:

     ```
     com.ibm.as400.access.AS400.proxyServer=http://myServer
     ```

7. Запустите программу-клиент.

Если вы хотите работать не только с классами Proxy, но и с другими классами, не входящими в jt400Proxy.jar, то вместо файла jt400Proxy.jar можно использовать файл jt400.jar. Файл jt400Proxy.jar представляет собой подмножество файла jt400.jar, т.е. все классы proxy содержатся в файле jt400.jar.

Примеры: Работа с серверами Proxy

Ниже приведены три примера выполнения описанных выше операций с поддержкой Proxy.

- Запуск приложения на Java с поддержкой Proxy
- Запуск аплета Java с поддержкой Proxy
- Запуск приложения на Java с поддержкой Proxy с туннелями.

Классы, поддерживающие сервер Proxy

Некоторые классы IBM Toolbox for Java могут работать с приложением сервера Proxy. Это следующие классы:

- JDBC
- Доступ на уровне записей
- Интегрированная файловая система
- Печать
- Очереди данных
- Вызов команд
- Вызов программ
- Вызов служебных программ
- Пользовательское пространство
- Область данных
- Класс AS400
- Класс SecureAS400

Другие классы в настоящее время jt400Proxy не поддерживает. Кроме того, опция прав доступа к интегрированной файловой системе в случае файла jt400Proxy.jar не работает. Однако включить нужные вам классы из файла jt400.jar можно с помощью класса JarMaker.

Пример: Запуск приложения на Java с поддержкой Proxy

В следующем примере показано, как запускать приложение на Java, используя поддержку Proxy.

1. Выберите компьютер, который будет работать как сервер Proxy. В переменной среды Java и в переменной CLASSPATH на сервере Proxy нужно указать файл `jt400.jar`. This machine must be able to connect to the system.
2. Запустите в этой системе сервер Proxy с помощью следующей команды: `java com.ibm.as400.access.ProxyServer -verbose` Подробный режим (verbose) позволяет отслеживать подключение и отключение клиентов от сервера.
3. Выберите компьютер, который будет работать как клиент. В переменной среды Java и в переменной CLASSPATH на сервере Proxy нужно указать файл `jt400.jar` и классы приложения. This machine must be able to connect to the proxy server but does not need a connection to the system.
4. Задайте в качестве значения системного свойства `com.ibm.as400.access.AS400.proxyServer` имя вашего сервера Proxy и запустите приложение. Это можно сделать с помощью опции -D большинства вызовов виртуальной машины Java: `java -Dcom.ibm.as400.access.AS400.proxyServer=Имя_Proxy-сервера Имя_приложения`
5. После запуска приложения вы должны увидеть (если на шаге 2 была задана опция verbose), что приложение установило по крайней мере одно соединение с сервером Proxy.

Пример: Запуск аплета Java с поддержкой Proxy

В следующем примере показано, как запускать аплет Java, используя поддержку Proxy.

1. Выберите компьютер, который будет работать как сервер Proxy. Аплеты могут инициализировать сетевые соединения только с тем компьютером, с которого они были первоначально загружены; поэтому

лучше всего запускать сервер Proxy на том же компьютере, на котором работает сервер HTTP. В переменной среды Java и в переменной CLASSPATH на сервере Proxy нужно указать файл jt400.jar.

2. Запустите в этой системе сервер Proxy с помощью следующей команды: `java com.ibm.as400.access.ProxyServer -verbose`. Подробный режим (опция verbose) позволяет отслеживать подключение и отключение клиентов от системы.
3. Перед запуском аплета сначала необходимо загрузить его программный код, поэтому постарайтесь уменьшить объем кода, насколько это возможно. Класс AS400ToolboxJarMaker позволяет значительно уменьшить размер файла jt400Proxy.jar, включив в него только код компонентов, применяемых аплетом. Например, если аплет использует только JDBC, то для уменьшения размера файла jt400Proxy.jar в него можно включить только часть кода минимального размера:

```
     java utilities.AS400ToolboxJarMaker
-source jt400Proxy.jar -destination jt400ProxySmall.jar
                                       -component JDBC
```

4. В аплете системному свойству com.ibm.as400.access.AS400.proxyServer необходимо присвоить имя сервера Proxy. Наиболее удобный способ сделать это - использовать для аплетов откомпилированный класс Properties (Пример). Откомпилируйте этот класс и поместите созданный файл Properties.class в каталог com/ibm/as400/access (в тот же каталог, в котором расположен файл html). Например, если файл .html - это /mystuff/HelloWorld.html, то файл Properties.class должен находиться в каталоге /mystuff/com/ibm/as400/access.
5. Поместите файл jt400ProxySmall.jar в каталог, в котором находится файл .html (/mystuff/ - см. шаг 4).
6. Создайте в вашем файле .html ссылку на аплет примерно следующего вида:

```
     <APPLET archive="jt400Proxy.jar,
Properties.class" code="YourApplet.class"
             width=300 height=100> </APPLET>
```

Пример: Запуск приложения на Java с поддержкой туннелей Proxy

В следующем примере показано, как запускать приложение на Java, используя поддержку туннелей Proxy.

1. Выберите сервер HTTP, на котором будет работать сервер Proxy, и настройте его для запуска сервлета com.ibm.as400.access.TunnelProxyServer (в файле jt400.jar). **Note:** Ensure that the HTTP server has a connection to the system that contains the data or resource that the application uses because the servlet connects to that system to carry out requests.
2. Выберите систему, которая будет выполнять роль клиента, и убедитесь, что переменная CLASSPATH в этой системе включает файл jt400Proxy.jar и файлы классов приложения. The client must be able to connect to the HTTP server but does not need a connection to the system.
3. Задайте в качестве значения свойства com.ibm.as400.access.AS400.proxyServer имя сервера HTTP в формате URL.
4. Run the application, setting the value of the com.ibm.as400.access.AS400.proxyServer property to be the name of your HTTP server in URL format. Проще всего сделать это, указав опцию -D при вызове виртуальной машины Java:

```
java -Dcom.ibm.as400.access.AS400.proxyServer=http://Имя_Proxy-сервера Имя_приложения
```

Примечание: Для создания правильного URL сервлета клиентское приложение Proxy объединяет слово servlet и имя сервлета в имя сервера. В этом примере http://Имя_Proxy-сервера преобразуется в http://Имя_Proxy-сервера/servlet/TunnelProxyServer

---

Сравнение Secure Sockets Layer и Java Secure Socket Extension

Продукт IBM Toolbox for Java поддерживает применение Java Secure Socket Extension (JSSE) в соединениях SSL Java. Продукт JSSE входит в состав продукта J2SE версии 1.4 и выше.

Дополнительные сведения о JSSE приведены на Web-сайте JSSE фирмы SUN  .

JSSE позволяет устанавливать защищенные соединения с идентификацией серверов и шифрованием передаваемой информации. С помощью JSSE можно организовать защищенный обмен данными между клиентами и серверами по любым протоколам стека TCP/IP (например, по HTTP или FTP).

Если ранее вы работали с sslight, перенесите приложения на JSSE. В V5R4 поддерживается только JSSE. sslight не поддерживается.

IBM Toolbox for Java 2 Micro Edition

The IBM Toolbox for Java 2 Micro Edition package (com.ibm.as400.micro) enables you to write Java programs that allow a variety of Tier0 wireless devices, like personal digital assistants (PDAs) and cell phones, to directly access System i5 data and resources.

ToolboxME requirements

Your workstation, wireless device, and server must meet certain requirements (listed below) for developing and running IBM Toolbox for Java 2 Micro Edition applications.

Although Toolbox ME is considered a part of IBM Toolbox for Java, it is not included in the licensed product. ToolboxME (jt400Micro.jar) is included in the open source version of Toolbox for Java, called JTOpen. You must separately download and set up ToolboxME, which is contained in JTOpen.

Требования

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

Требования к рабочей станции

Workstation requirements for developing ToolboxME applications:

+ • A supported version of Java Standard Edition
- • Java virtual machine for wireless devices
- • Симулятор или эмулятор беспроводного устройства

Требования к беспроводному устройству

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

Требования к серверу

Server requirements for using ToolboxME applications:

- Класс MEServer, который входит в IBM Toolbox for Java или в последнюю версию JTOpen
- Требования к i5/OS для работы с IBM Toolbox for Java

Downloading and setting up ToolboxME

You must separately download IBM Toolbox for Java 2 Micro Edition (jt400Micro.jar), which is contained in JTOpen.

You can download ToolboxME from the IBM Toolbox for Java/JTOpen Web site  that also offers additional information about setting up ToolboxME.

How you set up ToolboxME is different for the Tier0 device, the development workstation, and the server:

- Создайте приложение для беспроводного устройства (с помощью файла jt400Micro.jar), затем установите приложение согласно инструкциям производителя устройства.
- Make sure that the System i Host Servers are started on the server that contains the target data.

- Убедитесь, что системе, в которой вы собираетесь запустить MEServer, доступен файл jt400.jar.

Дополнительная информация приведена на следующих страницах:

“Установка IBM Toolbox for Java на рабочей станции” на стр. 10

“Installing IBM Toolbox for Java on your system” на стр. 9

Concepts important for using ToolboxME

Before you begin developing IBM Toolbox for Java 2 Micro Edition Java applications, you need to understand the following concepts and standards that govern such development.

Java 2 Platform, Micro Edition (J2ME)

J2ME - это реализация стандарта Java 2, предоставляющего среды выполнения Java для беспроводных устройств Tier0, таких как электронные записные книжки (PDA) и сотовые телефоны. IBM Toolbox for Java 2 Micro Edition отвечает этому стандарту.

Устройства Tier0

Устройствами Tier0 называются беспроводные устройства, такие как PDA и сотовые телефоны. Эти устройства подключаются к компьютерам и сетям по беспроводным соединениям. Название связано с обычной трехуровневой моделью приложений (tier - уровень). Трехуровневая модель описывает распределенную программу, подразделяющуюся на три основные части, расположенные на разных компьютерах или сетях:

- Третий уровень состоит из базы данных и связанных программ, находящихся на сервере; как правило, этот сервер не совпадает с сервером второго уровня. Третий уровень предоставляет информацию и способы доступа к ней остальным уровням.
- Второй уровень - это коммерческие и деловые приложения, обычно расположенные на другом компьютере (часто - сервере), подключенном к общей сети.
- Первый уровень - это часть приложения рабочей станции, включая пользовательский интерфейс.

Устройства Tier0 обычно невелики по размерам и массе и ограничены в ресурсах. Типичными примерами могут служить PDA и сотовые телефоны. Устройства Tier0 заменяют или дополняют возможности устройств первого уровня.

Конфигурация подключенных устройств с ограниченными ресурсами (CLDC)

Конфигурация определяет минимальный набор API и необходимых средств виртуальной машины Java, позволяющий предоставить ожидаемые функции большому количеству устройств. Конфигурация CLDC предназначена для широкого набора устройств с ограниченным набором ресурсов, в частности, для устройств Tier0.

За дополнительной информацией обратитесь к разделу CLDC  .

Профайл мобильных устройств (MIDP)

Профайл представляет набор API, основанный на существующий конфигурации. Профайл предназначен для работы устройств определенного типа или операционной системы. Профайл MIDP, основанный на конфигурации CLDC, предоставляет стандартную среду выполнения, позволяющую динамически развертывать приложения и службы для работы с устройствами Tier0.

Дополнительная информация приведена в разделе Mobile Information Device Profile (MIDP)  .

Виртуальная машина Java для беспроводных устройств

Для запуска приложения Java устройству Tier0 необходима виртуальная машина Java, специально разработанная с учетом ограниченности ресурсов беспроводного устройства. Ниже перечислены некоторые возможные JVM:

- IBM J9 virtual machine, part of the IBM WebSphere Micro Environment
- Виртуальная машина Sun K (KVM) в составе CLDC
- MIDP

Связанная информация

Для создания приложений Java, предназначенных для поддержки беспроводных устройств, вы можете воспользоваться любым из множества существующих средств разработки. For a brief list of such tools, see Related information for IBM Toolbox for Java.

Дополнительную информацию и загружаемые симуляторы и эмуляторы беспроводных устройств вы найдете на Web-сайте, посвященном устройству или операционной системе, для которой предназначено ваше приложение.

ToolboxME classes

The com.ibm.as400.micro package provides the classes necessary to write applications that enable your Tier0 devices to access server data and resources.

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component.

ToolboxME provides the following classes:

com.ibm.as400.micro package

“Устройства Tier0” на стр. 354

Класс MEServer

Use the IBM Toolbox for Java MEServer class to fulfill requests from your Tier0 client application that uses the IBM Toolbox for Java 2 Micro Edition jar file. От имени клиентского приложения класс MEServer создает объекты IBM Toolbox for Java и применяет к ним методы.

Примечание: To use ToolboxMe classes, you must separately download and set up the ToolboxME component. For more information, see Downloading and setting up ToolboxME.

Для запуска класса MEServer служит следующая команда:

```
java com.ibm.as400.micro.MEServer [опции]
```

где [опции] могут принимать следующие значения:

-pcml pcml_doc1 [;pcml_doc2;...]
Задает документ PCML для загрузки и синтаксического анализа. Сокращенное название этой опции: `-pc`.

For important information about using this option, see the MEServer Javadoc.

-port порт
Задает порт для установления соединений с клиентами. По умолчанию это порт 3470. Сокращенное название этой опции: `-po`.

-verbose [true|false]
Указывает, печатать ли информацию о состоянии и соединении в System.out. Сокращенное название этой опции: `-v`.

-help Печатает информацию о формате команды в System.out. Сокращенное название этой опции: `-h` или `-?`. По умолчанию информация о формате команды не печатается.

Запустить MEServer не удастся, если указанный порт уже занят другим сервером.

Информация, связанная с данной

MEServer Javadoc

“Устройства Tier0” на стр. 354

Класс AS400

The AS400 class in the micro package (com.ibm.as400.micro.AS400) provides a modified subset of the functions available in the AS400 class in the access package (com.ibm.as400.access.AS400). Use the IBM Toolbox for Java 2 Micro Edition AS400 class to sign on the system from a Tier0 device.

Примечание: To use ToolboxME classes, you must separately download and set up the ToolboxME component. For more information, see Downloading and setting up ToolboxME.

Класс AS400 позволяет:

- Connect to the MEServer
- Disconnect from the MEServer

Соединение с сервером MEServer устанавливается неявно. For example, after you create an AS400 object, you can use the run() method in CommandCall to automatically perform connect(). Иными словами, вам не требуется явно вызывать метод connect(), если только вы не хотите проконтролировать установление соединения.

Пример: Применение класса AS400

R The following example shows how to use the AS400 class to sign on to on the system:

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    system.connect();
}
catch (Exception e)
{
    // Обработка исключительной ситуации
}
// Обработка системного объекта завершена.
system.disconnect();
```

AS400 Javadoc

“Класс AS400” на стр. 21
The IBM Toolbox for Java AS400 class manages a set of socket connections to the server jobs on server and sign-on behavior for the server, including prompting the user for sign-on information, password caching, and default user management.

“Устройства Tier0” на стр. 354

“CommandCall class - Micro package” на стр. 357
The CommandCall class in the micro package (com.ibm.as400.micro.CommandCall) provides a modified subset of the functions available in the CommandCall class in the access package (com.ibm.as400.access.CommandCall). Use the CommandCall class to call an i5/OS command from a Tier0 device.

CommandCall class - Micro package

The CommandCall class in the micro package (com.ibm.as400.micro.CommandCall) provides a modified subset of the functions available in the CommandCall class in the access package (com.ibm.as400.access.CommandCall). Use the CommandCall class to call an i5/OS command from a Tier0 device.

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component.

The CommandCall run() method requires a String (the command you want to run) and returns any messages resulting from running the command as a String. Если команда выполняется без выдачи сообщений, то метод run() возвращает пустую строку.

Пример: применение объекта CommandCall

Ниже приведен пример применения класса CommandCall.

```
// Работа с командами.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    // Запуск команды "CRTLIB FRED."
    String[] messages = CommandCall.run(system, "CRTLIB FRED");
    if (messages != null)
    {
        // Необходимо сообщить об ошибке.
        System.out.println("Команда не выполнена:");
        for (int i = 0; i < messages.length; ++i)
        {
            System.out.println(messages[i]);
        }
    }
    else
    {
        System.out.println("Команда успешно выполнена!");
    }
}
catch (Exception e)
{
    // Обработка исключительной ситуации
}
// Обработка системного объекта завершена.
system.disconnect();
```

Ссылки, связанные с данной

“CommandCall - Access package” на стр. 29
The CommandCall class allows a Java program to call a non-interactive System i5 command.

Информация, связанная с данной

Micro package CommandCall Javadoc

“Устройства Tier0” на стр. 354

DataQueue class - Micro package

Use the DataQueue class to have your Tier0 device read from or write to a System i data queue. The DataQueue class in the micro package (com.ibm.as400.micro.DataQueue]) provides a modified subset of the functions available in the DataQueue class in the access package (com.ibm.as400.access.DataQueue).

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component.

В класс DataQueue входят следующие методы:

- Read or write an entry as a String

- Read or write an entry as an array of bytes

To read or write entries, you need to supply the name of the server where the data queue resides and the fully qualified integrated file system path name of the data queue. Если информации нет, то результатом чтения будет пустое значение.

Пример: Чтение и запись в очередь данных с помощью класса DataQueue

R The following example demonstrates how to use the DataQueue class to read entries from and write entries to a System
R i data queue:

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    // Запись в очередь данных.
    DataQueue.write(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ", "some text");

    // Чтение из очереди данных.
    String txt = DataQueue.read(system, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");
}
catch (Exception e)
{
    // Обработка исключительной ситуации
}
// Обработка системного объекта завершена.
system.disconnect();
```

Ссылки, связанные с данной

“Очереди данных” на стр. 42
Классы DataQueue позволяют программам на Java работать с очередями данных сервера.

“Concepts important for using ToolboxME” на стр. 354
Before you begin developing IBM Toolbox for Java 2 Micro Edition Java applications, you need to understand the following concepts and standards that govern such development.

Информация, связанная с данной

DataQueue Javadoc

“Устройства Tier0” на стр. 354

Micro package - ProgramCall class

Use the ProgramCall class to enable a Tier0 device to call an i5/OS program and access the data that is returned after the program runs. The ProgramCall class in the micro package (com.ibm.as400.micro.ProgramCall) provides a modified subset of the functions available in the ProgramCall class in the access package (com.ibm.as400.access.ProgramCall).

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component. Дополнительная информация приведена в разделе “ToolboxME requirements” на стр. 7.

To use the ProgramCall.run() method, you must provide the following parameters:

- Система, в которой следует запустить программу
- Имя документа “Язык описания вызовов программ” на стр. 381
- Имя запускаемой программы
- Хэш-таблица с именами задаваемых параметров программы и связанными с ними значениями
- Строковый массив с именами возвращаемых параметров

В классе ProgramCall входные и выходные параметры программы описываются на языке PCML. Файл PCML должен находиться на том же компьютере, что и MEServer, и каталог файла PCML должен быть указан в CLASSPATH этого компьютера.

Вы должны зарегистрировать все документы PCML на сервере MEServer. Для этого необходимо передать MEServer имя описанной в файле программы PCML, которую следует запустить. Вы можете зарегистрировать документ PCML во время выполнения или при запуске MEServer.

For more information about the hashtable that contains program parameters or how to register a PCML document, see the ToolboxME ProgramCall Javadoc. Дополнительная информация о PCML приведена в разделе “Язык описания вызовов программ” на стр. 381.

Пример: Применение объекта ProgramCall

The following example shows how to use the ProgramCall class to use your Tier 0 device to run a program on a server:

```
// Вызов программ.
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");

String pcmlName = "qsyrusri.pcml"; // Документ PCML с описанием нужной программы.
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);

    // Получение и просмотр пользовательского профайла.
    System.out.println("Пользовательский профайл: " + valuesToGet[0]);

    // Получение и просмотр даты в читаемом формате.
    char[] c = valuesToGet[1].toCharArray();
    System.out.println("Дата последнего входа в систему: " + c[3]+c[4]+"/"+c[5]+c[6]+"/"+c[1]+c[2] );

    // Получение и просмотр времени в читаемом формате.
    char[] d = valuesToGet[2].toCharArray();
    System.out.println("Время последнего входа в систему: " + d[0]+d[1]+":"+d[2]+d[3]);

    // Получение и просмотр информации о входе в систему.
    System.out.println("Информация о входе в систему: " + valuesToGet[3] );
}
catch (MEException te)
{
    // Обработка исключительной ситуации.
}
catch (IOException ioe)
{
    // Обработка исключительной ситуации
}

// Обработка системного объекта завершена.
system.disconnect();
```

ProgramCall Javadoc (micro package)

“Access package - ProgramCall class” на стр. 157
The IBM Toolbox for Java ProgramCall class allows the Java program to call a System i5 program. You can use the ProgramParameter class to specify input, output, and input/output parameters. If the program runs, the output and

input/output parameters contain the data that is returned by the System i5 program. If the System i5 program fails to run successfully, the Java program can retrieve any resulting System i5 messages as a list of AS400Message objects.

“Устройства Tier0” на стр. 354

Классы JdbcMe

The IBM Toolbox for Java 2 Micro Edition classes provide JDBC support, including support for the java.sql package. The classes are meant to be used in a program that runs on a Tier 0 device.

The following sections discuss accessing and using data and describe what is in JdbcMe.

Доступ к данным и их использование

Желательно, чтобы работа с данными на устройстве Tier0 ничем не отличалась от работы на обычном стационарном компьютере. Однако большая часть аппаратного и программного обеспечения устройств Tier0 ориентирована на синхронизацию данных. Синхронизация позволяет хранить на каждом устройстве Tier0 точную копию данных из главной базы данных. Время от времени пользователи синхронизируют информацию на своих устройствах с содержимым главной базы данных.

Синхронизация данных значительно затрудняется в случае, если данные динамические. При работе с динамическими данными их обновление должно происходить достаточно быстро. Длительное ожидание синхронизации таких данных, как правило, неприемлемо. Кроме того, к аппаратному и программному обеспечению серверов и устройств, отвечающих за синхронизацию данных, зачастую предъявляются достаточно высокие требования.

To help solve the problems inherent in the data synchronization model, the JdbcMe classes in ToolboxME enable you to perform live updates and access the main database, but still allow offline data storage. Приложение может обращаться к важным данным в автономной памяти и в то же время без промедления заносить информацию в главную базу данных. Такой компромиссный подход сочетает в себе достоинства синхронизации данных и режима прямого доступа.

Содержимое JdbcMe

В силу объективных причин любой драйвер для устройства Tier0 должен быть небольшим. Однако API JDBC очень велик по своему размеру. Классы JdbcMe очень малы, но поддерживают достаточное количество интерфейсов JDBC, что позволяет устройствам Tier0 выполнять нужные операции.

Классы JdbcMe предоставляют следующие функции JDBC:

- Вставка и обновление данных
- Управление транзакциями и изменение уровней изоляции транзакций
- Наборы результатов, допускающие прокрутку и обновление
- Поддержка SQL вызовов хранимых процедур и триггеров

Кроме того, классы JdbcMe обладают некоторыми уникальными особенностями:

- Универсальный драйвер, позволяющий объединить большинство параметров конфигурации в одном месте на сервере
- Стандартный механизм сохранения данных в автономной памяти

ToolboxME provides a java.sql package that follows the JDBC specification but contains only the smallest set of useful classes and methods. Благодаря этому классы JdbcMe имеют малый размер, при этом позволяя выполнять обычные задачи JDBC.

“Устройства Tier0” на стр. 354

Using ToolboxME to connect to a database on the host server:

The JdbcMeConnection class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCConnection class. Use JdbcMeConnection to enable your Tier0 device to access DB2 Universal Database (UDB) databases on the host server.

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component. For more information, see ToolboxME requirements and installation.

Use JdbcMeDriver.getConnection() to connect to the server database. В качестве параметров методу getConnection() передается строка URL, ИД пользователя и пароль. После этого администратор драйвера JDBC на сервере хоста пытается найти драйвер, который может подключиться к базе данных с заданным URL. Синтаксис строки URL для JdbcMeDriver следующий:

```
jdbc:as400://server-name/default-schema;meserver=<server>[:port];[other properties];
```

Примечание: Предыдущий пример синтаксиса занимает всего две строки, поэтому его можно быстро загрузить и напечатать. Как правило, URL занимает одну строку без пробелов или дополнительных пустых символов.

Вы должны указать имя сервера, в противном случае JdbcMeDriver выдаст исключительную ситуацию. Схема по умолчанию необязательна. Если вы не укажете порт, то JdbcMeDriver выберет порт 3470. Кроме того, вы можете задать некоторые свойства JDBC в строке URL. Синтаксис свойств следующий:

```
имя1=значение1;имя2=значение2;...
```

Полный список свойств, поддерживаемых JdbcMeDriver, приведен в разделе Свойства JDBC.

Пример: Подключение к серверу с помощью класса JdbcMeDriver

Пример: Подключение к базе данных сервера без указания схемы по умолчанию, номера порта и свойств JDBC

В этом примере ИД пользователя и пароль применяются в качестве параметров следующего метода:

```
        // Подключение к системе 'mysystem'. Схема по умолчанию, порт
// и свойства JDBC не указываются.
   Connection c  = JdbcMeDriver.getConnection("jdbc:as400://mysystem.helloworld.com;meserver=myMeServer;"
                                              "auser",
                                              "apassword");
```

Пример: Подключение к базе данных сервера с указанием схемы по умолчанию и свойств JDBC

В этом примере ИД пользователя и пароль применяются в качестве параметров следующего метода:

```
        // Подключение к системе 'mysystem'. Указываются схема и
// два свойства JDBC. Не указывается порт.
Connection c2 = JdbcMeDriver.getConnection(
      "jdbc:as400://mysystem.helloworld.com/mySchema;meserver=myMeServer;naming=system;errors=full;"
      "auser",
      "apassword");
```

Пример: Подключение к базе данных сервера

В этом примере свойства (включая ИД пользователя и пароль) задаются с помощью URL:

```
       // Подключение с указанием свойств. Свойства задаются в URL,
// а в не объекте свойств.
   Connection c = DriverManager.getConnection(
      "jdbc:as400://mySystem;meserver=myMeServer;naming=sql;errors=full;user=auser;password=apassword");
```

Пример: Отключение от базы данных

В этом примере подключенный объект отключается от сервера с помощью метода close():

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

JdbcMeConnection Javadoc

AS400JDBCConnection Javadoc

“Устройства Tier0” на стр. 354

Класс JdbcMeDriver:

Use JdbcMeDriver in your Tier0 client application to run simple SQL statements that have no parameters and obtain ResultSets that the statements produce. The JdbcMeDriver class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCStatement class.

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component. Дополнительная информация приведена в разделе “Downloading and setting up ToolboxME” на стр. 353.

Явная регистрация JdbcMeDriver не выполняется; драйвер определяется с помощью свойства **driver**, указываемого в URL в методе JdbcMeConnection.getConnection(). Например, для загрузки драйвера JDBC IBM Developer Kit for Java (его также называют ’внутренним’) можно воспользоваться таким кодом:

```
Connection c = JdbcMeDriver.getConnection(
   "jdbc:as400://mysystem.myworld.com;meserver=myMeSrvr;driver=native;user=auser;password=apassword");
```

В отличие от остальных классов IBM Developer Kit for Java, получающих информацию с сервера, драйвер JDBC IBM Developer Kit for Java не требует передачи объекта AS400 в качестве входного параметра. Однако объект AS400 используется во внутренних процедурах, поэтому вы должны явно указать ИД пользователя и пароль. Укажите ИД пользователя и пароль либо в строке URL, либо в виде параметров в методе getConnection().

Примеры применения метода getConnection() приведены в разделе JDBCMeConnection.

Ссылки, связанные с данной

“Наборы результатов”
The IBM Toolbox for Java 2 Micro Edition result set classes are JdbcMeLiveResultSet, JdbcMeOfflineResultSet, JdbcMeResultSetMetaData.

Информация, связанная с данной

JdbcMeDriver Javadoc

“Устройства Tier0” на стр. 354

Наборы результатов:

The IBM Toolbox for Java 2 Micro Edition result set classes are JdbcMeLiveResultSet, JdbcMeOfflineResultSet, JdbcMeResultSetMetaData.

Классы JdbcMeLiveResultSet и JdbcMeOfflineResultSet содержат одни и те же функции, за следующим исключением:

- Класс JdbcMeLiveResultSet получает данные путем отправки вызова в базу данных на сервере
- Класс JdbcMeOfflineResultSet получает данные из базы данных на локальном устройстве

Примечание: To use ToolboxME classes, you must separately download and set up the ToolboxME component. For more information, see Downloading and setting up ToolboxME.

JdbcMeLiveResultSet

The JdbcMeLiveResultSet class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCResultSet class. Класс JdbcMeLiveResultSet в клиентском приложении Tier0 позволяет получить доступ к таблице данных, созданной в результате выполнения запроса.

Класс JdbcMeLiveResultSet получает строки таблицы последовательно. В пределах строки вы можете обращаться к значениям столбцов в любом порядке. Методы класса JdbcMeLiveResultSet позволяют выполнить следующие действия:

- Retrieve data of various types that are stored in the result set
- Переместить курсор в указанную строку (предыдущую, текущую, следующую и т.п.)
- Insert, update, and delete rows
- Update columns (using String and int values)
- Retrieve the ResultSetMetaData object that describes the columns in the result set

Курсор - это внутренний указатель на строку таблицы результатов, к которой в данный момент обратилась программа на Java. JDBC 2.0 provides additional methods for accessing specific positions within a database. These are the available scrollable cursor positions:

- absolute
- first
- last
- moveToCurrentRow
- moveToInsertRow
- previous
- relative

Функция прокрутки

Записи таблицы результатов, созданной в результате выполнения оператора, можно просматривать (прокручивать) от начала к концу или от конца к началу.

Таблица результатов, позволяющая перемещаться по записям таким образом, называется таблицей с возможностью прокрутки. В таких таблицах различают абсолютную и относительную позицию курсора. Так, вы можете переместиться на некоторую строку таблицы, указав ее положение относительно текущей строки (относительную позицию курсора). Кроме того, вы можете перейти к строке, указав ее номер (абсолютную позицию курсора).

JDBC 2.0 предоставляет две дополнительные функции прокрутки, которые можно применять при работе с классом ResultSet: прокрутка без учета изменений и прокрутка с учетом изменений.

В отличие от таблицы результатов с возможностью прокрутки, в открытую таблицу результатов без возможности прокрутки нельзя внести изменения. Драйвер JDBC IBM Toolbox for Java не поддерживает таблицы результатов без возможности прокрутки.

Таблица результатов с возможностью обновления

В приложениях могут применяться таблицы результатов, доступные только для чтения (в данные нельзя вносить изменения) или для изменения (разрешено изменение данных; для управления доступом других транзакций к базе данных может применяться блокировка на запись). В таблице результатов с возможностью обновления можно изменять, вставлять и удалять строки.

See the method summary in the Javadoc for a complete listing of the update methods available in JdbcMeResultSet.

Пример: Таблицы результатов с возможностью обновления

Ниже приведен пример таблицы с возможностью обновления (update) и внесения изменений в открытую таблицу результатов (scroll sensitive).

```
          // Подключение к серверу.
     Connection c = JdbcMeDriver.getConnection(
        "jdbc:as400://mySystem;meserver=myMeServer;user=auser;password=apassword");

          // Создание объекта Statement.
Установка возможности обновления
          // для набора результатов.
     Statement s = c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

          // Запуск запроса. Результат запроса помещается
          // в объект ResultSet.
     ResultSet rs = s.executeQuery ("Выберите имя и ИД для обновления в MYLIBRARY.MYTABLE");

          // Просмотр строк таблицы результатов.
          // В каждой строке старый ИД заменяется на новый.
     int newId = 0;
     while (rs.next ())
     {

               // Получение значений из ResultSet. Первое значение - это
               // строка, второе - целое число.
          String name = rs.getString("Имя");
          int id = rs.getInt("ИД");

          System.out.println("Имя = " + name);
          System.out.println("Старый ИД = " + id);

               // Обновление целочисленного ИД.
          rs.updateInt("ID", ++newId);

               // Запись обновлений на сервер.
          rs.updateRow ();

          System.out.println("Новый ИД = " + newId);
     }

          // Закрытие Statement и Connection.
     s.close();
     c.close();
```

Класс JdbcMeOfflineResultSet

The JdbcMeOfflineResultSet class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCResultSet class. Use JdbcMeOfflineResultSet in in your Tier0 client application to access a table of data that is generated by running a query.

С помощью JdbcMeOfflineResultSet вы можете работать с данными, находящимися на устройстве Tier0. Это могут быть как данные, уже существующие на устройстве, так и данные, которые вы поместили туда с помощью метода JdbcMeStatement.executeToOfflineData(). Метод executeToOfflineData() загружает и сохраняет на устройстве все данные, удовлетворяющие критериям запроса. Впоследствии вы можете воспользоваться классом JdbcMeOfflineResultSet для доступа к сохраненным данным.

Методы класса JdbcMeOfflineResultSet позволяют выполнить следующие действия:

- Retrieve data of various types that are stored in the result set
- Переместить курсор в указанную строку (предыдущую, текущую, следующую и т.п.)
- Insert, update, and delete rows
- Update columns (using String and int values)
- Retrieve the ResultSetMetaData object that describes the columns in the result set

R You can provide the ability to synchronize the local device database with the database on the server by using the
R functions present in the JdbcMe classes.

Класс JdbcMeResultSetMetaData

The JdbcMeResultSetMetaData class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCResultSetMetaData class. Класс JdbcMeResultSetMetaData в клиентском приложении Tier0 позволяет определить типы и свойства столбцов в наборах результатов JdbcMeLiveResultSet и JdbcMeOfflineResultSet.

Ниже приведен пример использования класса JdbcMeResultSetMetaData:

```
     // Подключение к серверу.
Connection c = JdbcMeDriver.getConnection(
   "jdbc:as400://mySystem;meserver=myMeServer;user=auser;password=apassword");

     // Создание объекта Statement.
Statement s = c.createStatement();

     // Запуск запроса. Результат заносится в объект  ResultSet.
     JdbcMeLiveResultSet rs = s.executeQuery ("Выберите имя и ИД в MYLIBRARY.MYTABLE");

     // Просмотр строк таблицы результатов.
while (rs.next ())
{

         // Получение значений из ResultSet. Первое значение - это
         // строка, второе - целое число.
    String name = rs.getString("Имя");
    int id = rs.getInt("ИД");

    System.out.println("Имя = " + name);
    System.out.println("ИД = " + newId);
}

     // Закрытие Statement и Connection.
s.close();
c.close();
```

Информация, связанная с данной

JDBCMeLiveResultSet Javadoc

AS400JDBCResultSet Javadoc

JDBCMeResultSetMetaData Javadoc

AS400JDBCResultSetMetaData Javadoc

JDBCMeOfflineResultSet Javadoc

Класс JdbcMeOfflineData:

The JdbcMeOfflineData class is an offline data repository meant to be used on a Tier0 device. Хранилище данных не зависит от применяемого профайла и виртуальной машины Java.

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component. For more information, see Downloading and setting up ToolboxME.

Методы класса JdbcMeOfflineData позволяют выполнить следующие действия:

- Create an offline data repository
- Open an existing repository
- Get the number of records in the repository
- Get and delete individual records
- Update records
- Add a record to the end of the repository
- Close the repository

Пример применения класса JdbcMeOfflineData приведен в следующем разделе:

"Example: Using ToolboxME, MIDP, and IBM Toolbox for Java" на стр. 711

Ссылки, связанные с данной

"Concepts important for using ToolboxME" на стр. 354
Before you begin developing IBM Toolbox for Java 2 Micro Edition Java applications, you need to understand the following concepts and standards that govern such development.

Информация, связанная с данной

JdbcMeOfflineData Javadoc

Класс JdbcMeStatement:

Use JdbcMeStatement in your Tier0 client application to run simple SQL statements that have no parameters and obtain ResultSets that the statements produce. The JdbcMeStatement class provides a subset of functions available in the IBM Toolbox for Java AS400JDBCStatement class.

Примечание: To use IBM Toolbox for Java 2 Micro Edition classes, you must separately download and set up the ToolboxME component. For more information, see Downloading and setting up ToolboxME.

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

Ниже приведен пример работы с объектом JdbcMeStatement:

```
     // Подключение к серверу.
JdbcMeConnection c = JdbcMeDriver.getConnection(
  "jdbc:as400://mysystem.helloworld.com/mylibrary;naming=system;errors=full;meserver=myMeServer;" +
  "user=auser;password=apassword");

     // Создание объекта Statement.
JdbcMeStatement s = c.createStatement();

     // Запуск оператора SQL, создающего таблицу в базе данных.
s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

     // Запуск оператора SQL, вставляющего запись в эту таблицу.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('DAVE', 123)");

     // Запуск оператора SQL, вставляющего запись в эту таблицу.
s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('CINDY', 456)");

     // Запуск запроса SQL на выбор данных из таблицы.
JdbcMeLiveResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

     // Закрытие Statement и Connection.
s.close();
c.close();
```

Информация, связанная с данной

JdbcMeStatement Javadoc

AS400JDBCStatement Javadoc

Creating and running a ToolboxME program

This information will enable you to edit, compile, and run the example IBM Toolbox for Java 2 Micro Edition program.

You can also use this information as a general guide for creating, testing, and running the ToolboxME working examples and your own ToolboxME applications.

В программе, рассмотренной в примере, используется виртуальная машина K Virtual Machine (KVM). Программа позволяет выполнить любой запрос JDBC. Затем над результатами запроса можно выполнить операции JDBC (Следующий, Предыдущий, Закрыть, Фиксация и Откат).

Before you begin creating any of the ToolboxME examples, make sure that your environment meets the ToolboxME requirements.

Creating the ToolboxME example

To create the ToolboxME example program for your Tier0 device, complete the following steps:

1. Copy the Java code for the ToolboxME example, called JdbcDemo.java.
2. В текстовом редакторе или редакторе Java измените некоторые разделы кода, как указано в комментариям к программе, и сохраните файл под именем JdbcDemo.java.

   Примечание: Воспользуйтесь инструментом для разработки приложений для беспроводных устройств; это позволит упростить выполнение остальных этапов. Некоторые средства разработки позволяют компилировать, проверять и создавать программу за один прием, а затем автоматически запускать ее в эмуляторе.
3. Откомпилируйте файл JdbcDemo.java, предварительно убедившись, что вы задали указатель на файл .jar, содержащий классы KVM.
4. Проверьте исполняемый файл с помощью средства разработки приложений поддержки беспроводных устройств или команды предварительной проверки Java.
5. Присвойте исполняемому файлу подходящий тип в зависимости от операционной системы устройства Tier0. Например, в случае OS Palm создайте файл JdbcDemo.prc.
6. Протестируйте программу. Если вы установили эмулятор, вы можете протестировать программу и посмотреть, как она выглядит, с помощью эмулятора.

   Примечание: Если вы тестируете программу для беспроводных устройств и в системе не установлено средство для разработки таких приложений, убедитесь, что на устройстве установлена виртуальная машина Java или MIDP.

See ToolboxME concepts for related information about concepts, wireless application development tools, and emulators.

Running the ToolboxME example

To run the ToolboxME example program on your Tier0 device, complete the following tasks:

- Загрузите исполняемый файл на устройство Tier0 согласно инструкциям производителя устройства.
- Запустите сервер MEServer
- Запустите программу JdbcDemo на устройстве Tier0, щелкнув на значке JdbcDemo.

ToolboxME example: JdbcDemo.java

To create this example as a working IBM Toolbox for Java 2 Micro Edition program, you need to copy the following .java file into a text or Java editor, make a few changes, then compile it.

Для копирования исходного кода просто выделите мышью весь показанный ниже код Java, щелкните правой кнопкой мыши и выберите **Скопировать**. Для вставки кода в окно редактора создайте пустой документ в редакторе, щелкните в нем правой кнопкой мыши и выберите **Вставить**. Не забудьте сохранить новый документ под именем JdbcDemo.java.

Создав файл .java, вернитесь к инструкциям по созданию и запуску примера программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME example. This program demonstrates how your wireless
// device can connect to the server and use JDBC to perform work on a
// получить доступ к удаленной базе данных.
//
////////////////////////////////////////////////////////////////////////////

import java.sql.*;               // Интерфейсы SQL, предоставленные JdbcMe
import com.ibm.as400.micro.*;    // Реализация JdbcMe
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.microedition.io.*;  // Часть спецификации CLDC
import de.kawt.*;                // Часть спецификации CLDC

class DemoConstants
{
    // Эти константы в основном применяются демонстрационной
    // версией драйвера JDBC. Идентификаторы создателя
// приложений Jdbc и JDBC (http://www.palmos.com/dev)
    // зарезервированы для операционной системы palm.
    public static final int demoAppID    = 0x4a444243; // JDBC
    // dbCreator создается несколько иначе, чтобы
    // пользователь видел базу данных Palm отдельно
    // от приложения JdbcDemo.
    public static final int dbCreator    = 0x4a444231; // JDB1
    public static final int dbType       = 0x4a444231; // JDB1
}

/**
 * Небольшое окно конфигурации, в котором показаны
 * текущие соединения/операторы, применяемый URL,
 * ИД пользователя и пароль
 */
class ConfigurationDialog extends Dialog implements ActionListener
{
    TextField         data;
    ConfigurationDialog(Frame w)
    {
        super(w, "Конфигурация");

        // Показать/изменить текущий URL соединения
        data = new TextField(JdbcDemo.mainFrame.jdbcPanel.url);
        add("В центре", data);

        // Кнопка OK.
        Panel panel = new Panel();
        Button button = new Button("Ok");
        button.addActionListener(this);
        panel.add(button);
        add("Внизу", panel);
        pack();
    }

    public void actionPerformed(ActionEvent e)
    {
        JdbcDemo.mainFrame.jdbcPanel.url = data.getText();
        data = null;
        setVisible(false);
    }
}

/**
 * Небольшое окно конфигурации, в котором показаны
 * текущие соединения/операторы, применяемый URL,
 * ИД пользователя и пароль
```

```
 */
class MultiChoiceDialog extends Dialog implements ActionListener
{
    Choice            task;
    ActionListener    theListener;
    MultiChoiceDialog(Frame w, String title, String prompt, String choices[], ActionListener it)
    {
        super(w, title);
        theListener = it;

        // Показать/изменить текущий URL соединения
        Label txt = new Label(prompt);
        add("Слева", txt);
        task = new Choice();
        for (int i=0; i<choices.length; ++i)
        {
            task.add(choices[i]);
        }
        task.select(0);
        add("В центре", task);

        // Кнопка OK.
        Panel panel = new Panel();
        Button button = new Button("Ok");
        button.addActionListener(this);
        panel.add(button);
        button = new Button("Отмена");
        button.addActionListener(this);
        panel.add(button);
        add("Внизу", panel);
        pack();
    }

    /**
     *  Определение выполняемого действия.
     **/
    public void actionPerformed(ActionEvent e)
    {
        int choice = task.getSelectedIndex();
        setVisible(false);
        if (e.getActionCommand().equals("OK"))
        {
            if (theListener != null)
            {
                ActionEvent ev = new ActionEvent(this,
                                                 ActionEvent.ACTION_PERFORMED,
                                                 task.getItem(choice));
theListener.actionPerformed(ev);
            }
            task = null;
        }
        else
        {
            // Никаких действий не выполняется
        }
    }
}

/**
 * JdbcPanel - это главная панель приложения.
 * В ее верхней части показаны текущее соединение и
 * оператор.
 * Затем следует текстовое поле для ввода операторов SQL.
 * Далее расположено поле Результаты для вывода каждого
 * столбца данных или результатов.
 * Затем - список задач и кнопка 'Перейти', позволяющая
 * перейти к нужной задаче.
```

```
 */
class JdbcPanel extends Panel implements ActionListener
{
    public final static int TASK_EXIT         = 0;
    public final static int TASK_NEW          = 1;
    public final static int TASK_CLOSE        = 2;
    public final static int TASK_EXECUTE      = 3;
    public final static int TASK_PREV         = 4;
    public final static int TASK_NEXT         = 5;
    public final static int TASK_CONFIG       = 6;
    public final static int TASK_TOPALMDB     = 7;
    public final static int TASK_FROMPALMDB   = 8;
    public final static int TASK_SETAUTOCOMMIT= 9;
    public final static int TASK_SETISOLATION = 10;
    public final static int TASK_COMMIT       = 11;
    public final static int TASK_ROLLBACK     = 12;


    // Объекты JDBC.
    java.sql.Connection  connObject  = null;
    Statement            stmtObject  = null;
    ResultSet         rs = null;
    ResultSetMetaData    rsmd        = null;

    String        lastErr     = null;
    String        url         = null;
    Label         connection  = null;
    Label         statement   = null;
    TextField     sql  = null;
    List          data = null;
    final Choice  task;

    /**
     * Создание GUI.
     */
    public JdbcPanel()
    {
        // URL JDBC
        // Обязательно исправьте следующую строку, чтобы она правильно указывала
        // the MEServer and the server to which you want to connect.
        url = "jdbc:as400://mySystem;user=myUidl;password=myPwd;meserver=myMEServer;";

        Panel    p1left = new Panel();
        p1left.setLayout(new BorderLayout());
        connection = new Label("Нет");
        p1left.add("Слева", new Label("Соед:"));
        p1left.add("В центре", connection);

        Panel    p1right = new Panel();
        p1right.setLayout(new BorderLayout());
        statement = new Label("Нет");
        p1right.add("Слева", new Label("Операт:"));
        p1right.add("В центре", statement);

        Panel    p1 = new Panel();
        p1.setLayout(new GridLayout(1,2));
        p1.add(p1left);
        p1.add(p1right);

        Panel    p2 = new Panel();
        p2.setLayout(new BorderLayout());
        p2.add("Вверху", new Label("SQL:"));
        sql = new TextField(25);
        sql.setText("выбрать * из QIWS.QCUSTCDT"); // Запрос по умолчанию
        p2.add("В центре", sql);

        Panel    p3 = new Panel();
```

```
    p3.setLayout(new BorderLayout());
    data = new List();
    data.add("Нет результатов");
    p3.add("Вверху", new Label("Результаты:"));
    p3.add("В центре", data);

    Panel    p4 = new Panel();

    task = new Choice();
    task.add("Выход");             // TASK_EXIT
    task.add("Создать");              // TASK_NEW
    task.add("Закрыть");            // TASK_CLOSE
    task.add("Выполнить");          // TASK_EXECUTE
    task.add("Пред.");             // TASK_PREV
    task.add("След.");             // TASK_NEXT
    task.add("Настр.");           // TASK_CONFIGURE
    task.add("RS в PalmDB");     // TASK_TOPALMDB
    task.add("Запрос в PalmDB");     // TASK_FROMPALMDB
    task.add("Установить AutoCommit");   // TASK_SETAUTOCOMMIT
    task.add("Установить Isolation");    // TASK_SETISOLATION
    task.add("Фиксация");           // TASK_COMMIT
    task.add("Откат");         // TASK_ROLLBACK
    task.select(TASK_EXECUTE);  // Начать отсюда.
    p4.add("Слева", task);

    Button b = new Button("Перейти");
    b.addActionListener(this);
    p4.add("Справа", b);

    Panel prest = new Panel();
    prest.setLayout(new BorderLayout());
    prest.add("Вверху", p2);
    prest.add("В центре", p3);
    Panel pall = new Panel();
    pall.setLayout(new BorderLayout());
    pall.add("Вверху", p1);
    pall.add("В центре", prest);

    setLayout(new BorderLayout());
    add("В центре", pall);
    add("Внизу", p4);
}

/**
 * Выполнить действие в зависимости от задачи,
 * выбранной в списке в настоящее время.
 */
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() instanceof MultiChoiceDialog)
    {
        String   cmd = e.getActionCommand();
        processExtendedCommand(cmd);
        return;
    }

    switch (task.getSelectedIndex())
    {
    case TASK_EXIT:
        System.exit(0);
        break;
    case TASK_NEW:
        JdbcPanel.this.goNewItems();
        break;
    case TASK_PREV:
        JdbcPanel.this.goPrevRow();
        break;
```

```
    case TASK_NEXT:
        JdbcPanel.this.goNextRow();
        break;
    case TASK_EXECUTE:
        if (connObject == null || stmtObject == null)
            JdbcPanel.this.goNewItems();

        JdbcPanel.this.goExecute();
        break;
    case TASK_CONFIG:
        JdbcPanel.this.goConfigure();
        break;
    case TASK_CLOSE:
        JdbcPanel.this.goClose();
        break;
    case TASK_TOPALMDB:
        if (connObject == null || stmtObject == null)
            JdbcPanel.this.goNewItems();

        JdbcPanel.this.goResultsToPalmDB();
        break;
    case TASK_FROMPALMDB:
        JdbcPanel.this.goQueryFromPalmDB();
        break;
    case TASK_SETAUTOCOMMIT:
        JdbcPanel.this.goSetAutocommit();
        break;
    case TASK_SETISOLATION:
        JdbcPanel.this.goSetIsolation();
        break;
    case TASK_COMMIT:
        JdbcPanel.this.goTransact(true);
        break;
    case TASK_ROLLBACK:
        JdbcPanel.this.goTransact(false);
        break;

    default :
    {
            Dialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, "Ошибка", "Задача не реализована");
            dialog.show();
            dialog = null;
        }
    }
}

public void processExtendedCommand(String cmd)
{
    try
    {
        if (cmd.equals("true"))
        {
            connObject.setAutoCommit(true);
            return;
        }
        if (cmd.equals("false"))
        {
            connObject.setAutoCommit(false);
            return;
        }
        if (cmd.equals("read uncommitted"))
        {
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_READ_UNCOMMITTED);
            return;
        }
        if (cmd.equals("read committed"))
        {
```

```
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_READ_COMMITTED);
            return;
        }
        if (cmd.equals("repeatable read"))
        {
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_REPEATABLE_READ);
            return;
        }
        if (cmd.equals("serializable"))
        {
            connObject.setTransactionIsolation(java.sql.Connection.TRANSACTION_SERIALIZABLE);
            return;
        }
        throw new IllegalArgumentException("Недопустимая команда: " + cmd);
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
        return;
    }
}

/**
 * Выполнить фиксацию или откат
 */
public void goTransact(boolean commit)
{
    if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Пропустить",
                                                   "Соединение не выделено");
        dialog.show();
        dialog = null;
        return;
    }
    try
    {
        if (commit)
            connObject.commit();
        else
            connObject.rollback();
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
 * Предложить пользователю задать значение автом. фиксации
 * Фактические действия, выполненные методом actionPerformed
 * при вызове processExtendedCommand().
 */
public void goSetAutocommit()
{
    if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Пропустить",
                                                   "Соединение не выделено");
        dialog.show();
        dialog = null;
        return;
    }
    try
    {
```

```
        String currentValue;
        if (connObject.getAutoCommit())
            currentValue = "Текущее: true";
        else
            currentValue = "Текущее: false";

        Dialog dialog = new MultiChoiceDialog(JdbcDemo.mainFrame,
                                              "Задать автом. фиксацию",
                                              currentValue,
                                              new String[]{ "true", "false"},
                                              this);
        dialog.show();
        dialog = null;
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
 * Предложить пользователю задать уровень изоляции,
 * фактические действия, выполненные методом actionPerformed()
 * при вызове processExtendedCommand().
 */
public void goSetIsolation()
{
    if (connObject == null)
    {
        FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                   "Пропустить",
                                                   "Соединение не выделено");
        dialog.show();
        dialog = null;
        return;
    }
    try
    {
        int   level = connObject.getTransactionIsolation();
        String currentLevel;
        switch (level)
        {
        case java.sql.Connection.TRANSACTION_READ_UNCOMMITTED:
            currentLevel = "Текущее: read uncommitted";
            break;
        case java.sql.Connection.TRANSACTION_READ_COMMITTED:
            currentLevel = "Текущее: read committed";
            break;
        case java.sql.Connection.TRANSACTION_REPEATABLE_READ:
            currentLevel = "Текущее: repeatable read";
            break;
        case java.sql.Connection.TRANSACTION_SERIALIZABLE:
            currentLevel = "Текущее: serializable";
            break;
        default : {
                currentLevel = "error";
            }
        }
        Dialog dialog = new MultiChoiceDialog(JdbcDemo.mainFrame,
                                              "Задать уровень изоляции",
                                              currentLevel,
                                              new String[]{ "read uncommitted",
                                                            "read committed",
                                                            "repeatable read",
                                                            "serializable"},
                                              this);
        dialog.show();
```

```
            dialog = null;
        }
        catch (Exception e)
        {
            JdbcDemo.mainFrame.exceptionFeedback(e);
        }
    }

    /**
     * Создать новое соединение или оператор.
     * В настоящее время поддерживается только одно
     * соединение и один оператор.
     */
    public void goNewItems()
    {
        if (connObject != null || stmtObject != null)
        {
            FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                       "Пропустить",
                                                       "Соед./опер. уже выделен");
            dialog.show();
            dialog = null;
        }
        if (connObject == null)
        {
            try
            {
                connObject = DriverManager.getConnection(url);
                //connection.setText(Integer.toString(((JdbcMeConnection)connObject).getId()));
                connection.repaint();
            }
            catch (Exception e)
            {
                JdbcDemo.mainFrame.exceptionFeedback(e);
                return;
            }
        }
        if (stmtObject == null)
        {
            try
            {
                try
                {
                    stmtObject = connObject.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                                            ResultSet.CONCUR_READ_ONLY);
                }
                catch (Exception e)
                {
                    // Повторная попытка...  DB2 NT версии 6.1 не поддерживает
                    // наборы результатов, допускающие прокрутку, поэтому будем
                    // считать, что другие базы данных JDBC 2.0 также ее не поддерживают.
// Попытаемся создать другой оператор.
                    try
                    {
                        stmtObject = connObject.createStatement();
                    }
                    catch (Exception ex)
                    {
                        // Если и вторая попытка оказалась неудачной,
                        // вновь выдается первая исключительная ситуация.
// Как правило, она более содержательна.
                        throw e;
                    }
                    FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                               "Вторая попытка успешно выполнена",
                                                               "Таблица результатов без прокрутки");
                    dialog.show();
```

```
                dialog = null;
            }

            statement.repaint();
        }
        catch (Exception e)
        {
            JdbcDemo.mainFrame.exceptionFeedback(e);
            return;
        }
    }
}


/**
 *  Закрытие оператора и соединения.
 **/
public void goClose()
{
    // Закрытие оператора.
    if (stmtObject != null)
    {
        if (rs != null)
        {
            try
            {
                rs.close();

                        }
            catch (Exception e)
            {
            }
            rs = null;
            rsmd = null;
        }
        try
        {
            stmtObject.close();
        }
        catch (Exception e)
        {
        }
        stmtObject = null;
        statement.setText("Нет");
        statement.repaint();
    }

    // Закрытие соединения.
    if (connObject != null)
    {
        try
        {
            connObject.close();
        }
        catch (Exception e)
        {
        }
        connObject = null;
        connection.setText("Нет");
        connection.repaint();
    }
    data.removeAll();
    data.add("Нет результатов");
    data.repaint();
    sql.repaint();
    return;
}
```

```
/**
 * Вывод окна конфигурации.
 **/
public void goConfigure()
{
    // Обратите внимание, что в KAWT не поддерживаются модельные окна диалога,
    // и работа программы возможна только потому, что изменяемые данные (URL)
    // были созданы до открытия этого окна диалога; пользователю
    // недоступен главный фрейм, пока данное окно открыто в palm (т.е. все
    // окна диалога в KAWT - модальные).
    ConfigurationDialog dialog = new ConfigurationDialog(JdbcDemo.mainFrame);
    dialog.show();
    dialog = null;
}

/**
 *  Выполнение указанного запроса.
 **/
public void goExecute()
{
    // Получение текущего выбранного оператора.
    try
    {
        if (rs != null)
            rs.close();


        rs = null;
        rsmd = null;
        boolean results = stmtObject.execute(sql.getText());
        if (results)
        {
            rs = stmtObject.getResultSet();
            rsmd = rs.getMetaData();
            // Вывод первой строки
            goNextRow();
        }
        else
        {
            data.removeAll();
            data.add(stmtObject.getUpdateCount() + " строк обновлены");
            data.repaint();
        }
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
 *  Переход к следующей строке набора результатов.
 **/
public void goNextRow()
{
    try
    {
        if (rs == null || rsmd == null)
            return;

        int   count = rsmd.getColumnCount();
        int   i;
        data.removeAll();
        if (!rs.next())
            data.add("Конец данных");
```

```
        else
        {
            for (i=1; i>=count; ++i)
            {
                data.add(rs.getString(i));
            }
        }
        data.repaint();
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}


/**
 *  Переход к предыдущей строке набора результатов.
 **/
public void goPrevRow()
{
    try
    {
        if (rs == null || rsmd == null)
            return;

        int   count = rsmd.getColumnCount();
        int   i;
        data.removeAll();
        if (!rs.previous())
            data.add("Начало данных");
        else
        {
            for (i=1; i<=count; ++i)
            {
                data.add(rs.getString(i));
            }
        }
        data.repaint();
    }
    catch (Exception e)
    {
        JdbcDemo.mainFrame.exceptionFeedback(e);
    }
}

/**
 *  Выполнение запроса и сохранение результатов в базе данных локальных устройств
 **/
public void goResultsToPalmDB()
{
    try
    {
        if (stmtObject == null)
        {
            FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                       "Пропустить",
                                                       "Нет оператора");
            dialog.show();
            dialog = null;
            return;
        }

        boolean results =
           ((JdbcMeStatement)stmtObject).executeToOfflineData(sql.getText(),
                                                              "JdbcResultSet",
                                                              DemoConstants.dbCreator,
```

```
                                                                DemoConstants.dbType);
            if (!results)
            {
                FeedbackDialog dialog = new FeedbackDialog(JdbcDemo.mainFrame,
                                                           "Нет данных",
                                                           "Недопустимый запрос");
                dialog.show();
                dialog = null;
                return;
            }
            data.removeAll();
            data.add("Обновлены Palm DB 'JdbcResultSet'");
            data.repaint();
        }
        catch (Exception e)
        {
            JdbcDemo.mainFrame.exceptionFeedback(e);
        }
    }


    /**
     *  Выполнение запроса из базы данных, расположенной на устройстве palm.
     **/
    public void goQueryFromPalmDB()
    {
        try
        {
            if (rs != null)
            {
                rs.close();

                            rs = null;
            }
            rs = new JdbcMeOfflineResultSet ("JdbcResultSet",
                                             DemoConstants.dbCreator,
                                             DemoConstants.dbType);
            rsmd = rs.getMetaData();
            // Если необходимо выполнить отладку вывода, то
            // с помощью этого метода можно создать дамп
            // содержимого PalmDB, представленного набором
            // результатов (поскольку метод применяет System.out,
            // он наиболее эффективен в эмуляторе Palm при
            // отладке приложений).
            // ((JdbcMeOfflineResultSet)rs).dumpDB(true);

            // Вывод первой строки.
            goNextRow();
        }
        catch (SQLException e)
        {
            JdbcDemo.mainFrame.exceptionFeedback(e);
        }
    }
}

public class JdbcDemo extends Frame
{
    /** ActionListener, завершающий работу приложения. Требуется
* только один экземпляр, причем он может применяться повторно
     */
    private static ActionListener    exitActionListener = null;
    /**
     * Главное приложение в этом процессе.
     */
    static         JdbcDemo mainFrame = null;
```

```
    JdbcPanel      jdbcPanel = null;

    public static ActionListener getExitActionListener()
    {
        if (exitActionListener == null)
        {
            exitActionListener = new ActionListener()
            {
                public void actionPerformed(ActionEvent e)
                {
                    System.exit(0);
                }
            };
        }
        return exitActionListener;
    }

    /**
     *  Конструктор демонстрационной версии
     **/
    public JdbcDemo()
    {
        super("Демонстрационная версия JDBC");
        setLayout(new BorderLayout());

        jdbcPanel = new JdbcPanel();
        add("Center", jdbcPanel);

        addWindowListener(new WindowAdapter()
         {
             public void windowClosing(WindowEvent e)
             {
                 System.exit(0);
             }
         });
        setSize(200,300);
        pack();
    }

    public void exceptionFeedback(Exception e)
    {
        Dialog dialog = new FeedbackDialog(JdbcDemo.mainFrame, e);
        dialog.show();
        dialog = null;
    }

    /**
     * Главный метод.
     **/
    public static void main( String args[] )
    {
        try
        {
            mainFrame = new JdbcDemo();
            mainFrame.show();
            mainFrame.jdbcPanel.goConfigure();
        }
        catch (Exception e)
        {
            System.exit(1);
        }
    }
}
```

ToolboxME working examples

The following IBM Toolbox for Java 2 Micro Edition working examples illustrate ways to use ToolboxME with the Mobile Information Device Profile (MIDP).

Для просмотра выбранных исходных файлов или загрузки всех исходных файлов, необходимых для создания приложений поддержки беспроводных устройств, выберите одну из следующих ссылок:

"Example: Using ToolboxME, MIDP, and JDBC" на стр. 703

"Example: Using ToolboxME, MIDP, and IBM Toolbox for Java" на стр. 711

"Downloading the ToolboxME examples"

For more information about how to build a ToolboxME application, see "Creating and running a ToolboxME program" на стр. 366.

Дополнительная информация о MIDP приведена в разделе "Профайл мобильных устройств (MIDP)" на стр. 354.

Downloading the ToolboxME examples

To build the IBM Toolbox for Java 2 Micro Edition examples into working wireless applications, you need all the source files and additional instructions.

Для загрузки примеров и создания исполняемых файлов выполните следующие действия:

1. Загрузите исходные файлы (microsamples.zip).
2. Разверните microsamples.zip в каталоге, специально созданном для этой цели.
3. В разделе "Creating and running a ToolboxME program" на стр. 366 приведены указания по созданию примеров приложений для поддержки беспроводных устройств.

Перед тем, как приступить к компиляции исходных файлов и созданию исполняемых файлов для устройства Tier0, ознакомьтесь с дополнительной информацией, приведенной в следующих разделах:

- "ToolboxME requirements" на стр. 7
- "Downloading and setting up ToolboxME" на стр. 353

Компоненты XML

IBM Toolbox for Java содержит некоторые компоненты языка XML, включая анализатор XML.

Эти компоненты позволяют упростить выполнение таких задач, как:

- Создание графических интерфейсов пользователя
- Calling programs on your system and retrieving the results
- Specifying data formats on your system

Язык описания вызовов программ

Program Call Markup Language (PCML) is a tag language that helps you call server programs with less Java code.

PCML основан на Расширяемом языке описаний (XML), который применяется для описания входных и выходных параметров программ сервера. Теги PCML позволяют полностью описать программу сервера, вызываемую приложением на Java.

Примечание: Если вы применяете или планируете применять PCML, возможно, вам придется использовать расширяемый язык вызовов программ (XPCML). XPCML расширяет функциональные

возможности и сферу применения PCML благодаря поддержке схем XML. Дополнительная информация о компонентах XML IBM Toolbox for Java, включая XPCML, приведена в разделе Компоненты XML.

Основным преимуществом PCML является то, что он позволяет сократить объем кода. Обычно для подключения к системе, получения данных и их преобразования из объектов сервера в объекты IBM Toolbox for Java требуется вставлять дополнительные фрагменты кода. PCML позволяет автоматически обрабатывать вызовы программ сервера с помощью классов IBM Toolbox for Java. Объекты класса PCML создаются из тегов PCML. Они позволяют значительно сократить объем кода, который необходимо написать для вызова программы сервера из приложения.

Несмотря на то, что PCML был разработан для поддержки распределенных вызовов серверных программ с клиентских платформ Java, с его помощью можно вызывать серверные программы, которые выполняются в серверной среде.

Requirements for using PCML

The PCML component has the same workstation Java virtual machine requirements as the rest of the IBM Toolbox for Java.

In addition, in order to parse PCML at run-time, the CLASSPATH for the application must include an XML parser. Анализатор XML должен расширять класс org.apache.xerces.parsers.SAXParser.

Примечание: If you preserialize the PCML file, you do not need to include an XML parser in the application CLASSPATH to run the application.

Ссылки, связанные с данной

“Требования к рабочей станции IBM Toolbox for Java” на стр. 8
Убедитесь, что рабочая станция отвечает следующим требованиям.

“Анализатор XML и обработчик XSLT” на стр. 418
Для применения некоторых пакетов или функций IBM Toolbox for Java необходимо, чтобы в переменной CLASSPATH во время их выполнения был указан анализатор XML или обработчик XSLT.

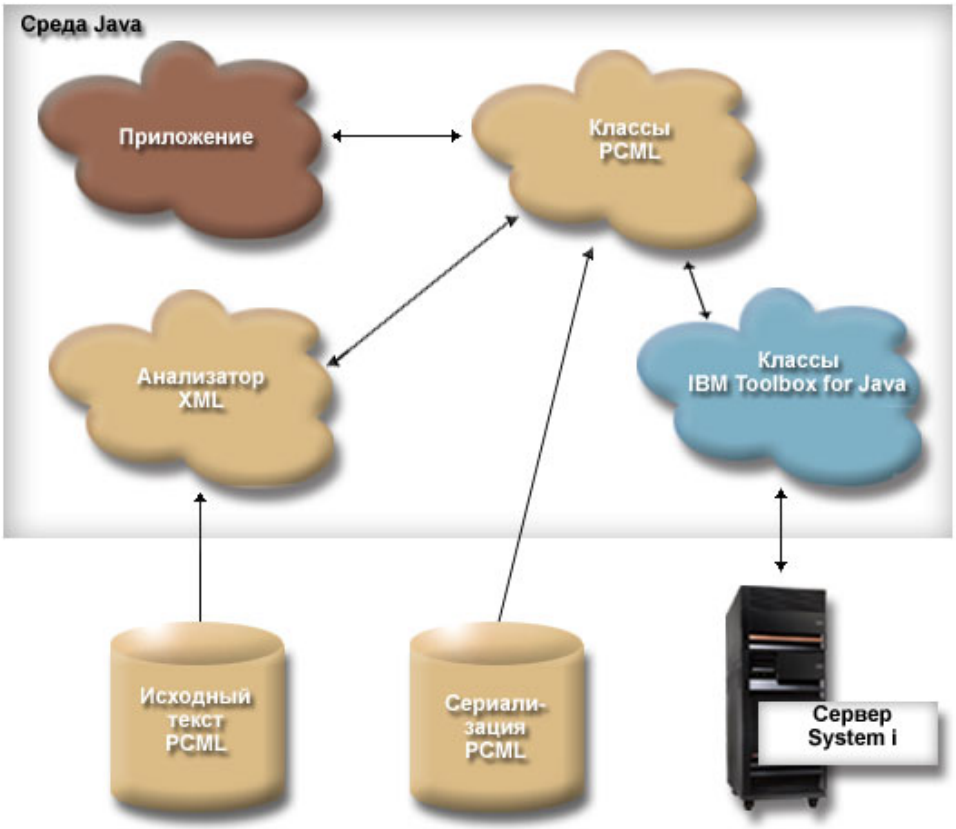
Building System i5 program calls with PCML

To build System i5 program calls with PCML, you must start by creating a Java application and a PCML source file.

R Depending on your design process, you must write one or more PCML source files where you describe the interfaces to
R the System i5 programs that will be called by your Java application. Подробное описание языка приведено в
R разделе Синтаксис PCML.

R Приложение на Java взаимодействует с классами PCML (в данном случае - с классом ProgramCallDocument).
R The ProgramCallDocument class uses your PCML source file to pass information between your Java application and
R the System i5 programs. Взаимодействие между приложениями на Java и классами PCML показано на рис. 1.

Рис. 1. Вызов программ сервера с помощью PCML.



Когда приложение создает объект ProgramCallDocument, синтаксический анализатор XML обрабатывает исходный файл PCML.Дополнительная информация о работе с анализатором XML в IBM Toolbox for Java приведена в разделе Анализатор XML и обработчик XSLT.

R After the ProgramCallDocument class has been created, the application program uses the ProgramCallDocument
R class's methods to retrieve the necessary information from the server through the System i5 distributed program call
R (DPC) server.

Для повышения производительности программы рекомендуется сохранить класс ProgramCallDocument в виде потока байтов во время компиляции продукта. Впоследствии класс ProgramCallDocument будет восстановлен из файла, содержащего этот поток байт. В этом случае анализатор XML не будет вызываться во время выполнения. Дополнительная информация приведена в разделе Работа с двоичными файлами PCML.

Работа с исходными файлами PCML

Во время создания объекта ProgramCallDocument приложение на Java обращается к исходному файлу PCML. Объект ProgramCallDocument рассматривает исходный файл PCML как ресурс Java. Приложение на Java определяет каталог исходного файла PCML с помощью переменной CLASSPATH

Ниже приведен фрагмент программы на Java, в котором создается объект ProgramCallDocument:

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "myPcmlDoc");
```

Объект ProgramCallDocument считывает исходный код PCML из файла myPcmlDoc.pcml. Обратите внимание, что в конструкторе не указано расширение .pcml.

Если приложение на Java создается в виде пакета Java, вы можете добавить имя пакета к имени ресурса PCML:

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.company.package.myPcmlDoc");
```

Работа с двоичными файлами PCML

Для повышения производительности программы можно воспользоваться двоичным файлом PCML. Такой файл PCML содержит сохраненные объекты Java, представляющие PCML. Это те объекты, которые были созданы вместе с объектом ProgramCallDocument на основе исходного файла, как это было описано выше.

Применение двоичных файлов PCML позволяет повысить производительность, так как в этом случае во время выполнения приложения не требуется вызывать анализатор XML для обработки тегов PCML.

Объекты PCML можно сохранить одним из следующих способов:

- Из командной строки:

  ```
  java com.ibm.as400.data.ProgramCallDocument -serialize mypcml
  ```

  Этот способ рекомендуется применять в том случае, когда компоновка приложения выполняется пакетным процессом.
- Из программы на Java:

  ```
  ProgramCallDocument pcmlDoc; // Инициализация
  pcmlDoc.serialize();
  ```

Если объекты PCML описаны в исходном файле myDoc.pcml, то они будут сохранены в файле myDoc.pcml.ser.

Сравнительный анализ исходных и двоичных файлов PCML

Рассмотрим следующий фрагмент программы, в котором создается объект ProgramCallDocument:

```
AS400 as400 = new AS400();
ProgramCallDocument pcmlDoc = new ProgramCallDocument(as400, "com.mycompany.mypackage.myPcmlDoc");
```

Сначала конструктор ProgramCallDocument попытается найти двоичный файл PCML с именем myPcmlDoc.pcml.ser в пакете com.mycompany.mypackage, просматривая каталоги, указанные в переменной CLASSPATH Java. Если двоичный файл PCML не существует, то конструктор попытается найти исходный файл PCML с именем myPcmlDoc.pcml в пакете com.mycompany.mypackage, просматривая каталоги, указанные в переменной CLASSPATH Java. Если исходный файл PCML не существует, будет вызвана исключительная ситуация.

Полные имена

R Your Java application uses ProgramCallDocument.setValue() to set input values for the System i5 program being
R called. Likewise, your application uses ProgramCallDocument.getValue() to retrieve output values from the System i5
R program.

When accessing values from the ProgramCallDocument class, you must specify the fully qualified name of the document element or **<data>** tag. Полное имя - это объединение имен всех внешних тегов, разделенных точками.

Например, для приведенного ниже исходного кода PCML полное имя элемента ″**nbrPolygons**″ - ″**polytest.parm1.nbrPolygons**″. Для получения координаты ″**x**″ одной из вершин многоугольника нужно указать имя ″**polytest.parm1.polygon.point.x**″.

Если элементу не присвоено имя, то для всех его потомков полное имя не определено. Программа на Java не может обращаться к элементам, у которых нет полного имени.

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Parameter 1 contains a count of polygons along with an array of polygons -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Each polygon contains a count of the number of points along with an array of points -->
      <struct name="polygon" count="nbrPolygons">
        <data name="nbrPoints" type="int" length="4" init="3" />
        <struct name="point" count="nbrPoints" >
          <data name="x" type="int" length="4" init="100" />
          <data name="y" type="int" length="4" init="200" />
        </struct>
      </struct>
    </struct>
  </program>
</pcml>
```

Обращение к элементам массива

Any **<data>** or **<struct>** element can be defined as an array using the **count** attribute. Or, a **<data>** or **<struct>** element can be contained within another **<struct>** element that is defined as an array.

Furthermore, a **<data>** or **<struct>** element can be in a multidimensional array if more than one containing element has a **count** attribute specified.

Для того чтобы приложение могло обращаться к массиву и его элементам, необходимо задать индекс для каждого измерения массива. Индексы массива передаются в виде массива значений типа **int**. Ниже приведен фрагмент программы на Java, иллюстрирующий работу с описанным выше массивом многоугольников:

```
ProgramCallDocument polytest; // Инициализация
Integer nbrPolygons, nbrPoints, pointX, pointY;
nbrPolygons = (Integer) polytest.getValue("polytest.parm1.nbrPolygons");
System.out.println("Число многоугольников:" + 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("  Число вершин:" + 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);
    }
}
```

Отладка

При работе со сложными структурами данных в коде PCML программисты часто допускают ошибки, которые приводят к возникновению исключительных ситуаций в классе ProgramCallDocument. Ошибки, связанные с неправильно заданным смещением или размером данных, очень нелегко находить и исправлять.

Включите трассировку PCML с помощью следующего метода класса Trace:

```
Trace.setTraceOn(true);      // Включение функции трассировки.
Trace.setTracePCMLOn(true);  // Включение трассировки PCML.
```

Примечание: Все общие методы класса PcmlMessageLog, включая трассировку, были отключены в выпуске V5R2.

Метод трассировки setFileName() позволяет отправлять в файлы протокола или, по умолчанию, в System.out, следующие типы данных:

- A dump of the hexadecimal data being transferred between the Java application and the System i5 program. Он содержит входные параметры программы, преобразованные в формат AS/400. Кроме того, он содержит выходные параметры, еще не преобразованные в формат Java.

  Данные записываются в обычном формате, в котором шестнадцатеричные значения расположены слева, а их символьная интерпретация - справа. Ниже приведен пример этого формата дампа. (Пример вывода был изменен для оптимального размещения по ширине)

  ```
  qgyolobj[6]
  Offset : 0....... 4....... 8....... C....... 0....... 4....... 8....... C.......
           0...4...8...C...0...4...8...C...
       0 : 5CE4E2D9 D7D9C640 4040
           **USRPRF                        *
  ```

  В приведенном выше примере в дампе показан седьмой параметр, содержащий 10 байт данных и равный ″*USRPRF″.

- Ниже приведен пример 16-ричного дампа, содержащего выходные параметры, и его интерпретация. (Пример вывода был изменен для оптимального размещения по ширине)

```
/QSYS.lib/QGY.lib/QGYOLOBJ.pgm[2]
Offset : 0....... 4....... 8....... C....... 0....... 4....... 8....... C.......
         0...4...8...C...0...4...8...C...
     0 : 0000000A 0000000A 00000001 00000068 D7F0F9F9 F0F1F1F5 F1F4F2F6 F2F5F400
         *................P09901151426254.*
    20 : 00000410 00000001 00000000 00000000 00000000 00000000 00000000 00000000
         *...............................*
    40 : 00000000 00000000 00000000 00000000
         *................                *
Reading data -- Offset: 0   Length: 4   Name: "qgyolobj.listInfo.totalRcds"
 Byte data: 0000000A
Reading data -- Offset: 4   Length: 4   Name: "qgyolobj.listInfo.rcdsReturned"
 Byte data: 0000000A
Reading data -- Offset: 8   Length: 4   Name: "qgyolobj.listInfo.rqsHandle"
 Byte data: 00000001
Reading data -- Offset: c   Length: 4   Name: "qgyolobj.listInfo.rcdLength"
 Byte data: 00000068
Reading data -- Offset: 10  Length: 1   Name: "qgyolobj.listInfo.infoComplete"
 Byte data: D7
Reading data -- Offset: 11  Length: 7   Name: "qgyolobj.listInfo.dateCreated"
 Byte data: F0F9F9F0F1F1F5
Reading data -- Offset: 18  Length: 6   Name: "qgyolobj.listInfo.timeCreated"
 Byte data: F1F4F2F6F2F5
Reading data -- Offset: 1e  Length: 1   Name: "qgyolobj.listInfo.listStatus"
 Byte data: F4
Reading data -- Offset: 1f  Length: 1   Name: "qgyolobj.listInfo.[8]"
 Byte data: 00
Reading data -- Offset: 20  Length: 4   Name: "qgyolobj.listInfo.lengthOfInfo"
 Byte data: 00000410
Reading data -- Offset: 24  Length: 4   Name: "qgyolobj.listInfo.firstRecord"
 Byte data: 00000001
Reading data -- Offset: 28  Length: 40  Name: "qgyolobj.listInfo.[11]"
 Byte data: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

The above messages can be very helpful in diagnosing cases where the output data coming from the System i5 program does not match the PCML source. Это часто происходит при работе с динамическими массивами данных и смещениями.

Синтаксис PCML

В языке PCML предусмотрены следующие теги, в свою очередь содержащие теги атрибутов.

- Тег программы открывает и закрывает описание одной программы

- Тег структуры определяет именованную структуру, которая может быть задана в качестве аргумента программы или элемента другой именованной структуры. Каждое поле структуры представляет собой тег данных или структуры.
- Тег данных определяет поле в программе или структуре.

Ниже приведен пример описания программы PCML, содержащей одну структуру и некоторые изолированные данные.

```
<program>

    <struct>
        <data> </data>
    </struct>

    <data> </data>

</program>
```

Тег программы на PCML:

Ниже приведен формат тега программы PCML.

```
<program name="имя"
    [ entrypoint="имя-точки-входа" ]
    [ epccsid="ccsid" ]
    [ path="полное-имя" ]
    [ parseorder="список-имен" ]
    [ returnvalue="{ void | integer }" ]
    [ threadsafe="{ true | false }" ]>
</program>
```

В следующей таблице перечислены атрибуты тега программы. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
entrypoint=	*имя-точки-входа*	Задает имя точки входа объекта вызываемой служебной программы.
epccsid=	*ccsid*	Задает CCSID точки входа объекта вызываемой служебной программы. Дополнительная информация приведена в примечаниях для служебных программ в разделе справочной документации Java для класса ServiceProgramCall.
name=	*имя*	Задает имя программы.

Атрибут	Значение	Описание
path=	*путь*	Задает путь к объекту программы. По умолчанию предполагается, что программа находится в библиотеке QSYS. Значением должен быть допустимый путь IFC к объекту *PGM или *SRVPGM. Если вызывается объект *SRVPGM, то должен быть задан атрибут вызываемой точки входа. Если атрибут точки входа не задан, то по умолчанию вызывается объект *PGM из библиотеки QSYS. Если атрибут точки входа задан, то по умолчанию вызывается объект *SRVPGM из библиотеки QSYS. Путь должен содержать только прописные символы. Атрибут **path** не следует применять, если путь в приложении задается во время работы, например, если пользователь задает библиотеку, применяемую при установке. В этом случае необходимо воспользоваться методом ProgramCallDocument.setPath().
parseorder=	*список-имен*	Задает порядок обработки выходных параметров. Значением должен быть список имен параметров, разделенных пробелами. Порядок имен определяет последовательность их обработки. The names in the list must be identical to the names specified on the **name** attribute of tags belonging to the **<program>**. По умолчанию выходные параметры обрабатываются в порядке появления тегов в документе. Некоторые программы возвращают в одном из параметров информацию о предыдущем параметре. Например, программа может возвращать в первом параметре массив структур, а во втором - размер этого массива. В этом случае второй параметр должен быть обработан первым, чтобы объект ProgramCallDocument ″знал″, сколько структур в первом параметре нужно обработать.
returnvalue=	*void* Программа не возвращает значение. *integer* Программа возвращает 4-байтовое значение со знаком.	Задает тип значения, возвращаемого служебной программой (если оно есть). Этот атрибут не поддерживается в вызовах объектов *PGM.

Атрибут	Значение	Описание
threadsafe=	*true* Программа поддерживает нити. *false* Программа не поддерживает нити.	When you call a Java program and an i5/OS program that are on the same server, use this property to specify whether you want to call the i5/OS program in the same job and on the same thread as the Java program. Если точно известно, что программа поддерживает работу с несколькими нитями, то для повышения производительности следует указать значение *true*. По умолчанию для защиты среды программы вызываются в различных заданиях сервера. Значение по умолчанию - *false*.

Тег структуры PCML:

Ниже приведен формат тега структуры PCML.

```
<struct name="имя"
    [ count={ число | имя-элемента-данных } ]
    [ maxvrm=строка-версии ]
    [ minvrm=строка-версии ]
    [ offset={ число | имя-элемента-данных } ]
    [ offsetfrom={ число | имя-элемента-данных | имя-структуры } ]
    [ outputsize={ число | имя-элемента-данных } ]
    [ usage="{ inherit | input | output | inputoutput }" ]>
</struct>
```

В следующей таблице перечислены атрибуты тега структуры. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
name=	*имя*	Specifies the name of the **<struct>** element
count=	*Число*, где *число* задает фиксированный неизменяемый массив с конечным числом элементов. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of elements in the array. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type="***int***"**. See Resolving Relative Names for more information about how relative names are resolved.	Указывает, что элемент является массивом указанного размера. Если атрибут count не задан, то элемент не является массивом, хотя он может быть элементом массива.

Атрибут	Значение	Описание
maxvrm=	*версия*	Задает максимальную версию системы i5/OS, в которой поддерживается этот элемент. Если версия i5/OS больше значения этого атрибута, то элемент и его потомки не будут обработаны в ходе вызова программы. Элемент **maxvrm** позволяет сглаживать различия в программных интерфейсах различных выпусков i5/OS. Версия должна быть задана в формате ″VvRrMm″, где ″V,″ ″R″ и ″M″ - это прописные буквы, а ″v,″ ″r″ и ″m″ задают версию, выпуск и модификацию, соответственно. Вместо ″v″ может быть указано значение от 1 до 255 включительно. Вместо ″r″ и ″m″ могут быть заданы значения от 0 до 255 включительно.
minvrm=	*версия*	Задает минимальную версию системы i5/OS, в которой поддерживается этот элемент. Если версия i5/OS больше значения этого атрибута, то элемент и его потомки не будут обработаны в ходе вызова программы. Этот атрибут позволяет сглаживать различия в программных интерфейсах в различных выпусках i5/OS. Версия должна быть задана в формате ″VvRrMm″, где ″V,″ ″R″ и ″M″ - это прописные буквы, а ″v,″ ″r″ и ″m″ задают версию, выпуск и модификацию, соответственно. Вместо ″v″ может быть указано значение от 1 до 255 включительно. Вместо ″r″ и ″m″ могут быть заданы значения от 0 до 255 включительно.

Атрибут	Значение	Описание
offset=	*Число*, где *число* определяет фиксированное неизменяемое смещение. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the offset to the element. **Имя-элемента-данных** может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=***″int″*. See Resolving Relative Names for more information about how relative names are resolved.	Specifies the offset to the **<struct>** element within an output parameter. Некоторые программы возвращают информацию в виде фиксированной структуры, за которой следует одно или несколько полей либо структур переменной длины. В этом случае расположение элемента переменной длины обычно задается в виде смещения в выходном параметре. The **offset** attribute is used to describe the offset to this **<struct>** element. Атрибут **Offset** указывается вместе с атрибутом **offsetfrom**. Если атрибут **offsetfrom** не задан, смещение, заданное в атрибуте **offset**, отсчитывается от родительского элемента. See Specifying Offsets for more information about how to use the **offset** and **offsetfrom** attributes. Атрибуты **offset** и **offsetfrom** применяются только для обработки вывода программы. Эти атрибуты не задают смещение ввода. Если этот атрибут не задан, элемент данных располагается в параметре сразу после предыдущего элемента, если он есть.

Атрибут	Значение	Описание
offsetfrom=	*Число*, где *число* определяет фиксированное неизменяемое основное расположение. Атрибут *число*, как правило, применяется для указания атрибута **number=***"0"*, означающего, что используется абсолютное смещение от начала параметра. *data-name* where *data-name* defines the name of a **<data>** element to be used as a base location for the offset. Заданный элемент данных должен быть предком исходного элемента. Смещение, заданное в атрибуте **offset**, будет отсчитываться от элемента данных, заданного в этом атрибуте. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае должно быть задано имя предка исходного элемента. See Resolving Relative Names for more information about how relative names are resolved. *struct-name* where *struct-name* defines the name of a **<struct>** element to be used as a base location for the offset. Заданный элемент данных должен быть предком исходного элемента. Смещение, заданное в атрибуте **offset**, будет отсчитываться от элемента данных, заданного в этом атрибуте. *Имя-структуры* может быть задано полностью или относительно текущего элемента. В любом случае должно быть задано имя предка исходного элемента. See Resolving Relative Names for more information about how relative names are resolved.	Задает точку отсчета смещения, заданного в атрибуте **offset**. Если атрибут **offsetfrom** не задан, смещение, заданное в атрибуте **offset**, отсчитывается от родительского элемента. See Specifying Offsets for more information about how to use the **offset** and **offsetfrom** attributes. Атрибуты **offset** и **offsetfrom** применяются только для обработки вывода программы. Эти атрибуты не задают смещение ввода.

Атрибут	Значение	Описание
outputsize=	*Число*, где *число* задает фиксированное неизменяемое число резервных байтов. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of bytes to reserve for output data. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information about how relative names are resolved.	Задает число байт, которое должно быть зарезервировано в выводе для данного элемента. Для выходных параметров переменной длины атрибут **outputsize** указывает, сколько байт должно быть зарезервировано для вывода программы сервера. Атрибут **Outputsize** может быть задан для любого поля или массива переменной длины, а также для всего параметра, содержащего одно или несколько полей переменной длины. Атрибут **Outputsize** необязателен. Его не нужно указывать для выходных параметров фиксированного размера. Значение атрибута задает общий размер элемента с учетом всех его дочерних элементов. Атрибут **outputsize** всех потомков игнорируется. If the attribute is omitted, the number of bytes to reserve for output data is determined at runtime by adding the number of bytes to reserve for all of the children of the **<struct>** element.
usage=	*inherit*	Назначение элемента наследуется от родительского элемента. Если у структуры нет предка, предполагается, что назначение равно **inputoutput**.
	input	Задает входное значение для программы хоста. Символьные и числовые значения преобразуются перед передачей программе.
	output	Задает выходное значение программы хоста. Символьные и числовые значения преобразуются перед передачей программе.
	inputoutput	Задает значение, которое одновременно является входным и выходным.

Выбор смещения

Некоторые программы возвращают информацию в виде фиксированной структуры, за которой следует одно или несколько полей либо структур переменной длины. В этом случае расположение элемента переменной длины обычно задается в виде смещения в параметре.

Абсолютное смещение - это расстояние от начала параметра до начала поля или структуры в байтах. Относительное смещение - это расстояние от начала другой структуры до начала другой структуры в байтах.

В случае абсолютного смещения нужно задать атрибут **offsetfrom=″0″**. Ниже приведен пример смещения от начала параметра:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains a path -->
    <struct name="receiver" usage="output" outputsize="2048">
      <data name="pathType"         type="int"  length="4" />
      <data name="offsetToPathName" type="int"  length="4" />
      <data name="lengthOfPathName" type="int"  length="4" />
      <data name="pathName"         type="char" length="lengthOfPathName"
                offset="offsetToPathName"  offsetfrom="0"/>
    </struct>
  </program>
</pcml>
```

В случае относительного смещения необходимо задать имя структуры, от начала которой отсчитывается смещение. Ниже приведен пример смещения от начала заданной структуры:

```
<pcml ="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains an object -->
    <struct name="receiver" usage="output" >
      <data name="objectName"       type="char"  length="10" />
      <data name="libraryName"      type="char"  length="10" />
      <data name="objectType"       type="char"  length="10" />
      <struct name="pathInfo" usage="output" outputsize="2048" >
        <data name="pathType"         type="int"  length="4" />
        <data name="offsetToPathName" type="int"  length="4" />
        <data name="lengthOfPathName" type="int"  length="4" />
        <data name="pathName"         type="char" length="lengthOfPathName"
                  offset="offsetToPathName"  offsetfrom="pathInfo"/>
      </struct>
    </struct>
  </program>
</pcml>
```

Тег данных PCML:

В теге данных PCML допустимы перечисленные ниже атрибуты.

Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута. Возможные значения атрибута представлены в виде списка, заключенного в фигурные скобки {}, и отделены друг от друга вертикальной чертой |. Для каждого атрибута можно задать только одно значение, которое должно быть указано без скобок.

```
<data type="{ char | int | packed | zoned | float | byte | struct }"
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT }"]
    [ ccsid={ число | имя-элемента-данных } ]
    [ chartype={ однобайтовый | двухбайтовый }]
    [ count={ число | имя-элемента-данных } ]
    [ init=строка ]
    [ length={ число | имя-элемента-данных } ]
    [ maxvrm=строка-версии ]
    [ minvrm=строка-версии ]
    [ name=имя ]
    [ offset={ число | имя-элемента-данных } ]
    [ offsetfrom={ число | имя-элемента-данных | имя-структуры } ]
    [ outputsize={ число | имя-элемента-данных | имя-структуры } ]
    [ passby= { ссылка | значение } ]
    [ precision=число ]
    [ struct=имя-структуры ]
    [ trim={ слева | справа | оба значения | ни одно из значений } ]
    [ usage="{ inherit | input | output | inputoutput }" ]>
</data>
```

В следующей таблице перечислены атрибуты тега данных. В ней указано имя атрибута, возможные значения и описание атрибута.

Атрибут	Значение	Описание
type=	*char*, где *char* соответствует символьному значению. Значение типа *char* возвращается в виде объекта *java.lang.String*. Дополнительная информация приведена в разделе Указание длины с помощью значений *char* . *int*, где *int* - это целое значение. Значение *int* возвращается в виде объекта *java.lang.Long*. Дополнительная информация приведена в разделе Указание длины и точности данных с помощью значений *int*. *packed*, где *packed* - это упакованное десятичное значение. Значение *packed* возвращается в виде объекта *java.math.BigDecimal*. Дополнительная информация приведена в разделе Указание длины и точности данных с помощью значений *packed*. *zoned*, где *zoned* - это зонное десятичное значение. Значение *zoned* возвращается в виде объекта *java.math.BigDecimal*. Дополнительная информация приведена в разделе Указание длины и точности данных с помощью значений *zoned*. *float*, где *float* - это значение с плавающей точкой. Атрибут **length** задает число байт (4 или 8). 4-байтовое целое значение возвращается в виде объекта *java.lang.Float*. В виде объекта *java.lang.Double* возвращается 8-байтовое целое значение. Дополнительная информация приведена в разделе Указание длины данных с помощью значений *float* . *byte*, где *byte* - это однобайтовое значение. Данные не преобразуются. Значение *byte* возвращается в виде массива значений *byte* (*byte[]*). См. раздел Указание длины данных с помощью значений *byte* . *struct* where *struct* specifies the name of the **<struct>** element. Объект *struct* позволяет определить структуру и многократно использовать ее в документе. Тег **type=***"struct"* равносилен вставке указанной структуры в документ. Тип *struct* не позволяет задавать длину данных и не поддерживает значение точности.	Задает тип данных (символьный, целочисленный, упакованный, зонный, с плавающей точкой, байтовый или структура). У разных типов данных атрибуты длины и точности принимают разные значения. Дополнительная информация приведена в разделе Значения длины и точности.

Атрибут	Значение	Описание
bidistringtype=	*DEFAULT*, где DEFAULT - строчный тип по умолчанию для однонаправленных данных (LTR). *ST4*, где *ST4* - это строчный тип 4. *ST5*, где *ST5* - это строчный тип 5. *ST6*, где *ST6* - это строчный тип 6. *ST7*, где *ST7* - это строчный тип 7. *ST8*, где *ST8* - это строчный тип 8. *ST9*, где *ST9* - это строчный тип 9. *ST10*, где *ST10* - это строчный тип 10. *ST11*, где *ST11* - это строчный тип 11.	Specifies the bidirectional string type for **<data>** elements with **type=***"char"*. Если этот атрибут не задан, то тип строки определяется с помощью явно заданного CCSID или CCSID хоста по умолчанию. Строчные типы определены в документации по Java для класса BidiStringType.
ccsid=	*Число*, где *число* определяет фиксированный неизменяемый CCSID. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя объекта, который будет во время выполнения программы содержать CCSID символьных данных. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information about how relative names are resolved.	Specifies the host Coded Character Set ID (CCSID) for character data for the **<data>** element. The **ccsid** attribute can be specified only for **<data>** elements with **type=***"char"*. Если этот атрибут не задан, то считается, что CCSID символьных данных совпадает с CCSID хоста по умолчанию.
chartype=	*onebyte*, где значение *onebyte* задает размер каждого символа. *twobyte*, где значение *twobyte* задает размер каждого символа. Если применяется *chartype*, то атрибут **length=***number* задает число символов, а не количество байт.	Задает размер каждого символа.
count=	*number*, где *number* определяет фиксированное неизменяемое число элементов конечного массива. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of elements in the array. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information about how relative names are resolved.	Указывает, что элемент является массивом указанного размера. Если атрибут *count* не задан, то элемент не является массивом, хотя он может быть элементом массива.

Атрибут	Значение	Описание
init=	*строка*	Specifies an initial value for the **<data>** element. The *init* value is used if an initial value is not explicitly set by the application program when **<data>** elements with **usage=***″input″* or **usage=***″inputoutput″* are used. Начальное значение применяется для инициализации скалярных значений. Если элемент представляет массив или содержится в структуре, определяющей массив, то указанным значением инициализируются все записи массива.
length=	*Число*, где *число* определяет количество байт, необходимых для задания данных. Если применяется атрибут *chartype*, то атрибут *number* задает число символов, а не количество байт. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the length. A *data-name* can be specified only for **<data>** elements with **type=***″char″* or **type=***″byte″*. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=***″int″*. See Resolving Relative Names for more information about how relative names are resolved.	Задает длину элемента данных. Назначение этого атрибута зависит от типа данных. Дополнительная информация приведена в разделе Значения длины и точности.
maxvrm=	*версия*	Specifies the highest version of i5/OS on which this element exists. If the i5/OS version is greater than the version specified on this attribute, this element and its children, if any exist, will not be processed during a call to a program. This attribute is helpful for defining program interfaces which differ between releases of i5/OS. Версия должна быть задана в формате ″VvRrMm″, где ″V,″ ″R″ и ″M″ - это прописные буквы, а ″v,″ ″r″ и ″m″ задают версию, выпуск и модификацию, соответственно. Вместо ″v″ может быть указано значение от 1 до 255 включительно. Вместо ″r″ и ″m″ могут быть заданы значения от 0 до 255 включительно.

Атрибут	Значение	Описание
minvrm=	*версия*	Specifies the lowest version of i5/OS on which this element exists. If the i5/OS version is less than the version specified on this attribute, this element and its children, if any exist, will not be processed during a call to a program. This attribute is helpful for defining program interfaces which differ between releases of i5/OS. Версия должна быть задана в формате ″VvRrMm″, где ″V,″ ″R″ и ″M″ - это прописные буквы, а ″v,″ ″r″ и ″m″ задают версию, выпуск и модификацию, соответственно. Вместо ″v″ может быть указано значение от 1 до 255 включительно. Вместо ″r″ и ″m″ могут быть заданы значения от 0 до 255 включительно.
name=	*имя*	Specifies the name of the **<data>** element.
offset=	*Число*, где *число* определяет фиксированное неизменяемое смещение. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the offset to this element. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=***″int″*. See Resolving Relative Names for more information about how relative names are resolved.	Specifies the offset to the **<data>** element within an output parameter. Некоторые программы возвращают информацию в виде фиксированной структуры, за которой следует одно или несколько полей либо структур переменной длины. В этом случае расположение элемента переменной длины обычно задается в виде смещения в параметре. Атрибут **offset** указывается вместе с атрибутом **offsetfrom**. Если атрибут **offsetfrom** не задан, смещение, заданное в атрибуте **offset**, отсчитывается от родительского элемента. Дополнительная информация о применении атрибутов **offset** и **offsetfrom** приведена в разделе Установка смещений. Атрибуты **offset** и **offsetfrom** применяются только для обработки вывода программы. Эти атрибуты не задают смещение ввода. Если этот атрибут не задан, элемент данных располагается в параметре сразу после предыдущего элемента, если он есть.

Атрибут	Значение	Описание
offsetfrom=	*Число*, где *число* определяет фиксированное неизменяемое расположение. *Число*, как правило, применяется для задания атрибута **number=***"0"*, указывающего на то, что используется абсолютное смещение от начала параметра. *data-name* where *data-name* defines the name of a **<data>** element used as a base location for the offset. Заданный элемент данных должен быть предком исходного элемента. Смещение, заданное в атрибуте **offset**, будет отсчитываться от элемента данных, заданного в этом атрибуте. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае должно быть задано имя предка исходного элемента. See Resolving Relative Names for more information about how relative names are resolved. *struct-name* where *struct-name* defines the name of a **<struct>** element used as a base location for the offset. Заданный элемент данных должен быть предком исходного элемента. Смещение, заданное в атрибуте **offset**, будет отсчитываться от элемента данных, заданного в этом атрибуте. *Имя-структуры* может быть задано полностью или относительно текущего элемента. В любом случае должно быть задано имя предка исходного элемента. See Resolving Relative Names for more information about how relative names are resolved.	Задает точку отсчета смещения, заданного в атрибуте **offset**. Если атрибут **offsetfrom** не задан, смещение, заданное в атрибуте **offset**, отсчитывается от родительского элемента. See Specifying Offsets for more information about how to use the **offset** and **offsetfrom** attributes. Атрибуты **offset** и **offsetfrom** применяются только для обработки вывода программы. Эти атрибуты не задают смещение ввода.

Атрибут	Значение	Описание
outputsize=	*Число*, где *число* задает фиксированное неизменяемое число резервных байтов. *data-name* where *data-name* defines the name of a **<data>** element within the PCML document that will contain, at runtime, the number of bytes to reserve for output data. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=***"int"*. See Resolving Relative Names for more information about how relative names are resolved.	Задает число байт, которое должно быть зарезервировано в выводе для данного элемента. For output parameters which are variable in length, the **outputsize** attribute is needed to specify how many bytes must be reserved for data to be returned from the i5/OS program. Атрибут **outputsize** может быть задан для любого поля или массива переменной длины, а также для всего параметра, содержащего одно или несколько полей переменной длины. Атрибут **Outputsize** необязателен. Его не нужно указывать для выходных параметров фиксированного размера. Значение атрибута задает общий размер элемента с учетом всех его дочерних элементов. Атрибут **outputsize** всех потомков игнорируется. If **outputsize** is omitted, the number of bytes to reserve for output data is determined at runtime by adding the number of bytes to reserve for all of the children of the **<struct>** element.
passby=	*Ссылка*, где значение *ссылка* говорит о том, что параметр передается по ссылке. При вызове программы ей будет передан указатель на значение параметра. *Значение*, где *значение* соответствует целому числу. Оно является допустимым, только если указаны атрибуты **type=** *int* и **length=***4*.	Задает способ передачи параметра: по ссылке или по значению. This attribute is allowed only when this element is a child of a **<program>** element defining a service program call.
precision=	*число*	Задает число значимых разрядов для некоторых числовых типов данных. Дополнительная информация приведена в разделе Значения длины и точности.
struct=	*имя*	Specifies the name of a **<struct>** element for the **<data>** element. A **struct** attribute can be specified only for **<data>** elements with **type=***"struct"*.

Атрибут	Значение	Описание
trim=	*Справа*, где *справа* - это значение по умолчанию, при котором все конечные пробелы будут усечены. Значение *слева*, которое соответствует усечению начальных пробелов. *Оба значения*, при выборе которого будут усечены как начальные, так и конечные пробелы. *Ни одно из значений*, при применении которого пробелы не будут усечены.	Задает способ усечения пробелов в символьных данных.
usage=	*inherit*	Назначение элемента наследуется от родительского элемента. Если у структуры нет предка, предполагается, что назначение равно *inputoutput*.
	input	Задает входное значение для программы хоста. Символьные и числовые значения преобразуются перед передачей программе.
	output	Задает выходное значение программы хоста. Символьные и числовые значения преобразуются перед передачей программе.
	inputoutput	Задает значение, которое одновременно является входным и выходным.

Выбор смещения

Некоторые программы возвращают информацию в виде фиксированной структуры, за которой следует одно или несколько полей либо структур переменной длины. В этом случае расположение элемента переменной длины обычно задается в виде смещения в параметре.

Абсолютное смещение - это расстояние от начала параметра до начала поля или структуры в байтах. Относительное смещение - это расстояние от начала другой структуры до начала другой структуры в байтах.

В случае абсолютного смещения нужно задать атрибут **offsetfrom="0"**. Ниже приведен пример смещения от начала параметра:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains a path -->
    <struct name="receiver" usage="output" outputsize="2048">
      <data name="pathType"         type="int"  length="4" />
      <data name="offsetToPathName" type="int"  length="4" />
      <data name="lengthOfPathName" type="int"  length="4" />
      <data name="pathName"         type="char" length="lengthOfPathName"
                offset="offsetToPathName"  offsetfrom="0"/>
    </struct>
  </program>
</pcml>
```

В случае относительного смещения необходимо задать имя структуры, от начала которой отсчитывается смещение. Ниже приведен пример смещения от начала заданной структуры:

```
<pcml version="1.0">
  <program name="myprog" path="/QSYS.lib/MYLIB.lib/MYPROG.pgm">
    <!-- receiver variable contains an object -->
    <struct name="receiver" usage="output" >
      <data name="objectName"       type="char"  length="10" />
      <data name="libraryName"      type="char"  length="10" />
      <data name="objectType"       type="char"  length="10" />
      <struct name="pathInfo" usage="output" outputsize="2048" >
        <data name="pathType"         type="int"  length="4" />
        <data name="offsetToPathName" type="int"  length="4" />
        <data name="lengthOfPathName" type="int"  length="4" />
        <data name="pathName"         type="char" length="lengthOfPathName"
                  offset="offsetToPathName"  offsetfrom="pathInfo"/>
      </struct>
    </struct>
  </program>
</pcml>
```

Значения длины и точности:

У разных типов данных атрибуты длины и точности различны.

В следующей таблице перечислены все типы данных с описанием возможных значений длины и точности.

Тип данных	Длина	Точность
type="char"	Размер элемента данных в байтах, который не обязательно равен числу символов. Вы должны указать *число* или *имя-данных*.	Неприменимо
type="int"	Размер элемента данных в байтах: 2, 4 или 8. Вы должны указать *число*.	Задает точность целого числа в битах и указывает, есть ли у него знак: • Для значения **length**="2" – Укажите значение **precision**="15" для целых двухбайтовых чисел со знаком. Это значение по умолчанию – Укажите значение **precision**="16" для целых двухбайтовых чисел без знака • Для значения **length**="4" – Укажите значение **precision**="31" для целых четырехбайтовых чисел со знаком. – Укажите значение **precision**="32" для целых четырехбайтовых чисел без знака • Для значения **length**="8" укажите **precision**="63" для целых восьмибитовых чисел со знаком
type="packed" или "zoned"	Число цифр в элементе данных. Вы должны указать *число*.	Число десятичных цифр в элементе. Допустимы значения от нуля до общего числа цифр, заданного в атрибуте **length**.
type="float"	Задает длину элемента данных в байтах, 4 или 8. Вы должны указать *число*.	Неприменимо

Тип данных	Длина	Точность
type="byte"	Число байтов в элементе данных. Вы должны указать *число* или *имя-данных*.	Неприменимо
type="struct"	Запрещено.	Неприменимо

Преобразование относительных имен

В качестве значения некоторых атрибутов можно задать имя другого элемента документа, или тега. Это имя может быть указано относительно текущего тега.

Сначала выполняется поиск относительного имени среди имен дочерних тегов. Если имя не найдено, происходит переход к родительскому тегу и операция повторяется. This resolution must eventually result in a match of a tag that is contained by either the **<pcml>** tag or the **<rfml>** tag, in which case the name is considered to be an absolute name, not a relative name.

Ниже приведен пример применения PCML:

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Parameter 1 contains a count of polygons along with an array of polygons -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Each polygon contains a count of the number of points along with an array of points -->
      <struct name="polygon" count="nbrPolygons">
        <data name="nbrPoints" type="int" length="4" init="3" />
        <struct name="point" count="nbrPoints" >
          <data name="x" type="int" length="4" init="100" />
          <data name="y" type="int" length="4" init="200" />
        </struct>
      </struct>
    </struct>
  </program>
</pcml>
```

Ниже приведен пример применения RFML:

```
<rfml version="4.0">
  <struct name="polygon">
    <!-- Каждый многоугольник содержит число вершин и массив их координат -->
-->
    <data name="nbrPoints" type="int" length="4" init="3" />
    <data name="point" type="struct" struct="point" count="nbrPoints" />
  </struct>
  <struct name="point" >
    <data name="x" type="int" length="4" init="100" />
    <data name="y" type="int" length="4" init="200" />
  </struct>
  <recordformat name="polytest">
    <!-- This format contains a count of polygons along with an array of polygons -->
    <data name="nbrPolygons" type="int" length="4" init="5" />
    <data name="polygon" type="struct" struct="polygon" count="nbrPolygons" />
  </recordformat>
</rfml>
```

Graphical Toolbox и PDML

The Graphical Toolbox, a set of UI tools, enables you to create custom user interface panels in Java.

L You can incorporate the panels into your Java applications, applets, or System i Navigator plug-ins. The panels may contain data obtained from the system, or data obtained from another source such as a file in the local file system or a program on the network.

GUI Builder - это визуальный редактор типа WYSIWYG, предназначенный для создания окон диалога, окон свойств и мастеров на языке Java. С помощью этого приложения вы можете добавлять, переносить и изменять управляющие элементы пользовательского интерфейса, расположенные на панели, а также просматривать панель целиком. Созданные определения панелей могут применяться в окнах диалога, окнах свойств и мастерах. Кроме того, их можно добавлять к разделенным панелям, составным панелям и панелям с закладками. Помимо этого, GUI Builder позволяет создавать определения меню, панелей инструментов и контекстных меню. Вы можете добавить объекты JavaHelp в описание панелей, в том числе контекстную справку.

Resource Script Converter преобразует описания ресурсов Windows в формат XML, который применяется в программах на Java . С помощью этой программы вы можете обрабатывать описания ресурсов Windows (файлы .rc) окон диалога и меню Windows. Преобразованные файлы можно затем отредактировать с помощью GUI Builder. Resource Script Converter может применяться в сочетании с GUI Builder для создания окон свойств и мастеров на основе файлов .rc.

Оба рассмотренных средства суть реализация новой технологии, называемой **Язык описаний определений панелей** (**PDML**). Язык PDML основан на Расширяемом языке описаний (XML), не зависит от платформы и предназначен для описания макета элементов пользовательского интерфейса. Определив панели с помощью PDML, вы можете просматривать их с помощью API выполнения, предусмотренного в Graphical Toolbox. При выводе панелей этот API интерпретирует описание PDML и отображает элементы пользовательского интерфейса с помощью классов Java Foundation.

Примечание: Для применения PDML необходима среда выполнения Java (JRE) версии 1.4 или выше.

Преимущества Graphical Toolbox

Сокращает объем кода и ускоряет разработку

Graphical Toolbox значительно ускоряет и упрощает создание пользовательских интерфейсов на языке Java. GUI Builder позволяет контролировать все параметры размещения элементов пользовательского интерфейса в панелях. Поскольку макет описывается на языке PDML, нет необходимости определять интерфейс путем написания кода на языке Java, как и повторно компилировать код в случае изменений. Как следствие, создание и обслуживание приложений на Java занимает значительно меньше времени. С помощью Resource Script Converter вы можете быстро преобразовать большое количество панелей Windows в формат Java.

Создание справки

Определение пользовательских интерфейсов на языке PDML дает и другие преимущества. Вся информация о панели записывается на формальном языке описаний. Это позволяет расширить возможности инструментов и предоставить разработчикам дополнительные функции. Например, и в GUI Builder, и в Resource Script Converter предусмотрена функция создания шаблонов электронной справки по панели в формате HTML. Вам потребуется всего лишь выбрать разделы справки, и они будут автоматически созданы. В шаблон справки автоматически добавляются ссылки на разделы справки. Это означает, что разработчик должен предоставить только саму справочную информацию. Среда выполнения Graphical Toolbox автоматически выдает нужный раздел справки по запросу пользователя.

Автоматическое объединение интерфейса и программного кода

В PDML предусмотрены теги, позволяющие связать управляющий элемент с определенным атрибутом компонента Javabean. После того как вы зададите классы компонентов, содержащие данные для панели, и свяжете их атрибуты с управляющими элементами, соответствующие инструменты смогут автоматически создать шаблон исходного кода на Java для этих компонентов. Во время выполнения Graphical Toolbox управляет обменом данными между указанными компонентами и управляющими элементами панели.

Независимость от платформы

Среда выполнения Graphical Toolbox поддерживает обработку событий, проверку пользовательских данных и стандартные способы обмена данными между управляющими элементами панели. Параметры пользовательского интерфейса на конкретной платформе устанавливаются

автоматически в зависимости от текущей операционной системы. С помощью программы GUI Builder вы можете посмотреть, как этот интерфейс будет выглядеть на различных платформах.

Graphical Toolbox содержит два инструмента создания пользовательского интерфейса. Программа GUI Builder позволяет быстро создавать новые панели в визуальной среде, а программа Resource Script Converter - преобразовывать существующие панели Windows в формат Java. Преобразованные файлы можно отредактировать с помощью GUI Builder. Оба инструмента поставляются на национальном языке.

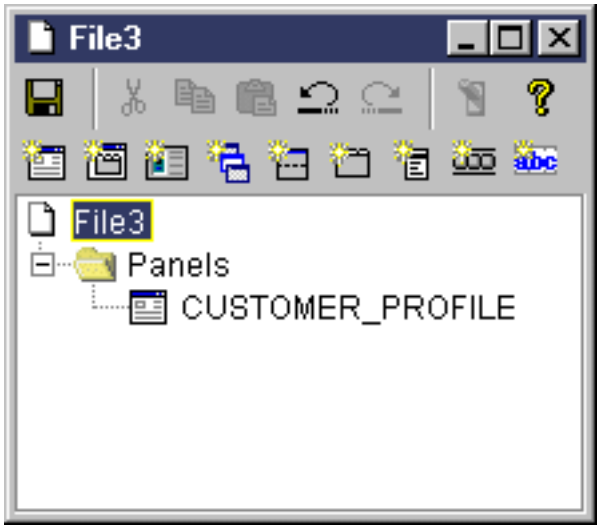
GUI Builder

При запуске программы GUI Builder появляются два окна, показанные на рис. 1:

Рис. 1: Окна GUI Builder

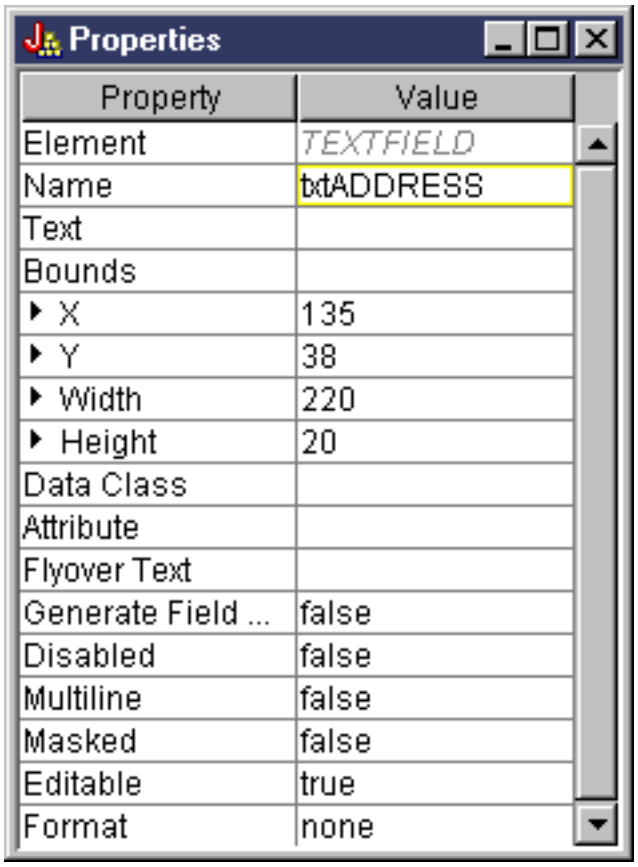
Создание и редактирование файлов PDML с помощью окна Редактор файлов.

Рис. 2: Окно Редактор файлов



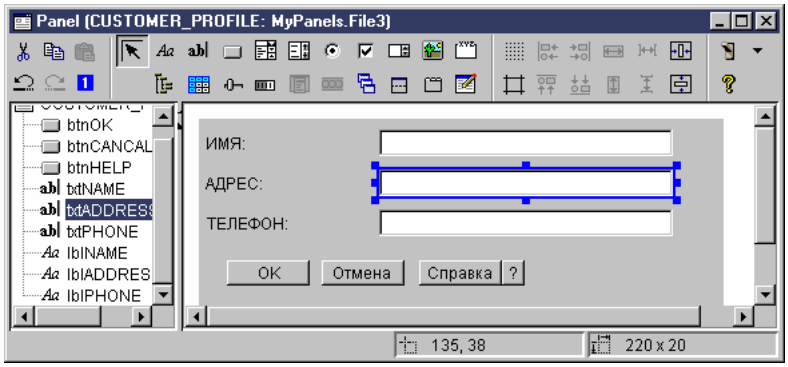
Окно Свойства предназначено для просмотра и изменения свойств выделенного управляющего элемента.

Рис. 3: Окно Свойства



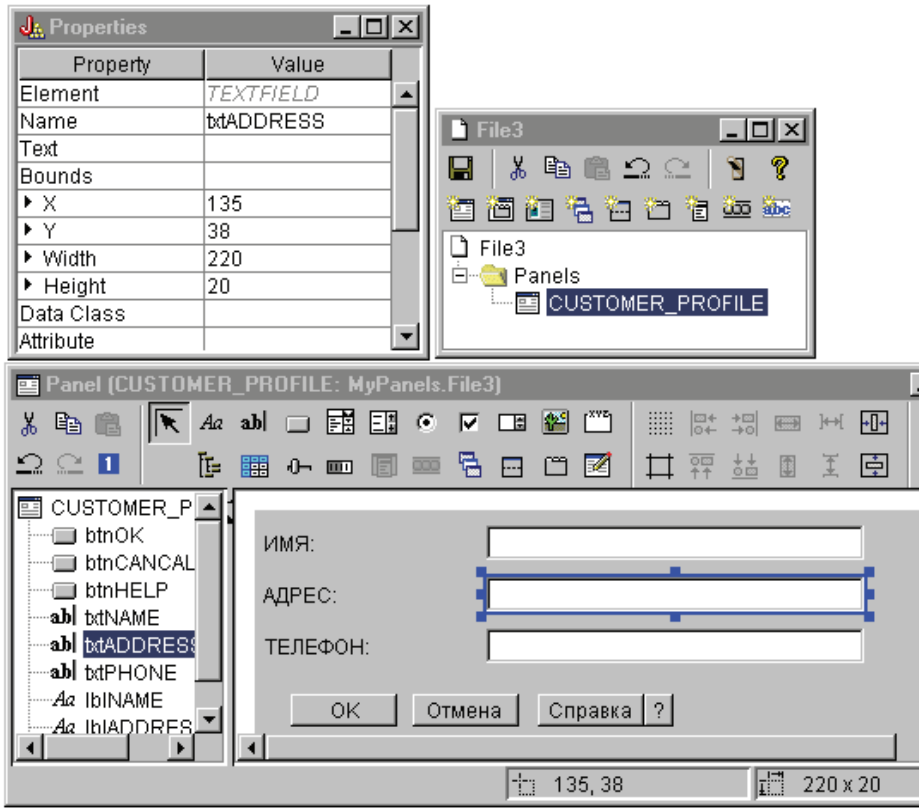
Создание и редактирование компонентов GUI с помощью окна Редактор панелей. Выберите компонент на панели и щелкните мышью в той точке панели, где вы хотите его разместить. На панели инструментов расположены значки для выравнивания группы элементов, предварительного просмотра панели и вызова электронной справки по функциям GUI Builder. Описание назначения каждого значка приведено в разделе Панель инструментов Редактора панелей GUI Builder.

Рис. 4: Окно Редактор панелей



В окне Редактор панелей всегда показана текущая панель, с которой вы работаете. На рис. 5 показана группа окон:

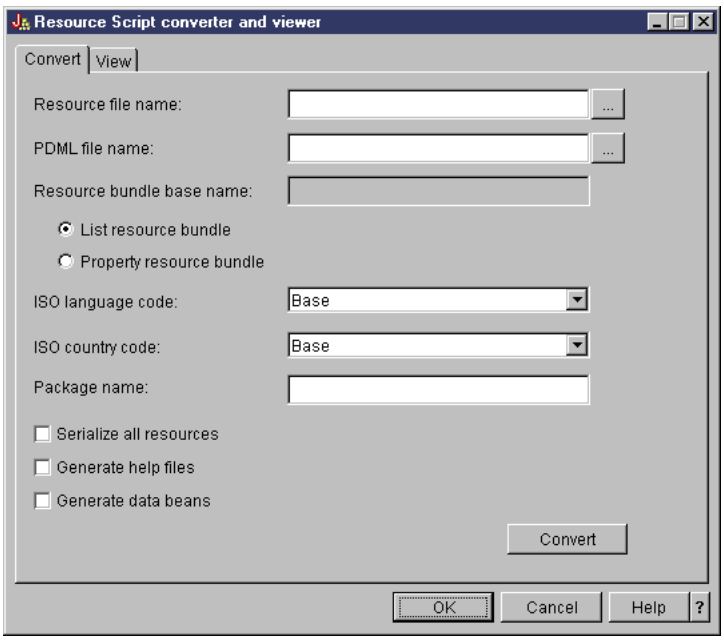
Рис. 5: Группа окон GUI Builder



Resource Script Converter

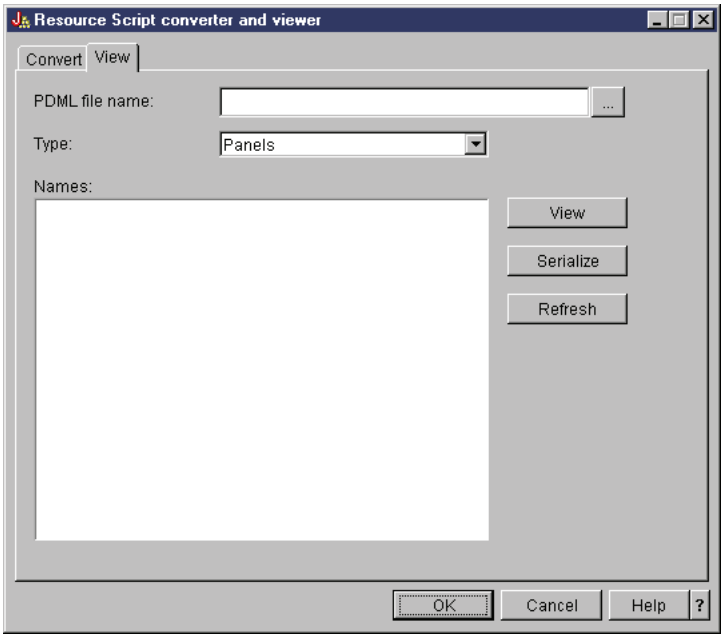
Окно программы Resource Script Converter - это окно диалога с закладками, состоящее из двух панелей. На панели **Преобразовать** вы должны указать имя файла .rc в формате Microsoft или VisualAge for Windows, который необходимо преобразовать в формат PDML. Кроме того, вы можете задать имя целевого файла PDML, а также набор ресурсов Java, который будет содержать преобразованное определение панели. Здесь же вы можете создать шаблон электронной справки по панели, создать шаблон исходного кода Java для объектов с данными для панели и сохранить определение панели в двоичном виде с целью повышения производительности. Подробное описание всех полей панели Преобразовать вы найдете в электронной справке по этой панели.

Рис. 6: Окно программы Resource Script Converter: Панель Преобразовать



После завершения преобразования с помощью панели **Вид** можно просмотреть содержимое созданного файла PDML и новые панели Java. При необходимости вы сможете внести небольшие изменения в панель с помощью программы GUI Builder. Перед преобразованием панели программа Resource Script Converter убеждается в отсутствии файла PDML с описанием панели и сохраняет все изменения для преобразования панели в будущем.

Рис. 7: Окно программы Resource Script Converter: Панель Показать



Язык описаний форматов записей (RFML)

RFML - это расширение XML, предназначенное для описания форматов записей.

Компонент RFML продукта IBM Toolbox for Java позволяет приложениям на Java использовать документы RFML для описания полей некоторых типов записей.

R RFML documents, called RFML source files, represent a useful subset of the data description specification (DDS) data
R types defined for System i physical and logical files. Документы RFML применяются для управления
R информацией в следующих объектах:

- Записи файлов
- Записи очередей данных
- Пользовательское пространство
- Буферах данных

Примечание: Дополнительная информация об описании атрибутов данных с помощью DDS приведена в разделе Справочник по DDS.

RFML очень похож на Язык описаний вызовов программ (PCML), другое расширение XML, которое поддерживается IBM IBM Toolbox for Java. RFML не является подмножеством языка PCML и не включает его в себя. Его можно назвать языком того же уровня, который содержит некоторые дополнительные элементы и атрибуты, но лишен некоторых элементов и атрибутов, присущих PCML.

PCML предоставляет основанную на XML технологию, служащую альтернативой классам ProgramCall и ProgramParameter. Аналогично, RFML служит более удобной альтернативой классам Record, RecordFormat и FieldDescription.

Информация, связанная с данной

Спецификации описания данных

Требования для применения RFML

Компонент RFML предъявляет те же требования к виртуальной машине Java рабочей станции, что и прочие компоненты IBM Toolbox for Java.

In addition, in order to parse RFML at run time, the CLASSPATH for the application must include an XML parser. Анализатор XML должен расширять класс org.apache.xerces.parsers.SAXParser. For more information, see“Анализатор XML и обработчик XSLT” на стр. 418.

Примечание: Требования к анализатору имен RFML совпадают с аналогичными требованиями PCML. Как и в случае с PCML, если вы заранее преобразуете файл RFML в двоичный формат, то анализатор XML не обязательно указывать в переменной CLASSPATH приложения.

Ссылки, связанные с данной

“Требования к рабочей станции IBM Toolbox for Java” на стр. 8
Убедитесь, что рабочая станция отвечает следующим требованиям.

Пример: Сравнение языка RFML с классами Record продукта IBM Toolbox for Java

Этот пример демонстрирует различия в применении языка RFML и классов Record продукта IBM Toolbox for Java.

Применение традиционных классов Record позволяет совместить спецификации формата данных с описанием алгоритма работы приложения. Для добавления, изменения или удаления поля требуется изменить и заново откомпилировать приложение на Java. Применение RFML дает возможность разместить спецификации формата данных в исходных файлах RFML, хранящихся отдельно от исходного кода приложения. Для изменения поля требуется изменить файл RFML. Обычно для этого не нужно изменять и заново компилировать приложение на Java.

В примере рассматривается приложение, работающее с записями заказчиков, заданными в исходном файле RFML с именем qcustcdt.rfml. В исходном файле определены поля, из которых состоят записи заказчиков.

В приведенном ниже списке указано, каким образом приложение на Java может проинтерпретировать записи заказчиков с помощью таких классов IBM Toolbox for Java, как Record, RecordFormat и FieldDescription:

```
// Буфер, содержащий двоичное представление одной записи.
    byte[] bytes;

    // ... Чтение данных записи в буфер...

    // Создание объекта RecordFormat для представления одной записи заказчика.
    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"));

    // Чтение из буфера байтов в объект RecordFormatDocument.
    Record rec1 = new Record(recFmt1, bytes);

    // Получение значений полей.
    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"));
```

Для сравнения ниже приведен способ интерпретации такой же записи с помощью RFML.

Код Java, интерпретирующий содержимое записи данных заказчика с помощью RFML имеет следующий вид:

```
    // Буфер, содержащий двоичное представление одной записи данных.
    byte[] bytes;

    // ... Чтение данных записи в буфер...

    // Преобразование файла RFML в объект RecordFormatDocument.
    // Имя исходного файла RFML - qcustcdt.rfml.
    RecordFormatDocument rfml1 = new RecordFormatDocument("qcustcdt");

    // Чтение из буфера байтов в объект RecordFormatDocument.
    rfml1.setValues("cusrec", bytes);

    // Получение значений полей.
    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"));
```

Класс RecordFormatDocument

Класс RecordFormatDocument позволяет преобразовывать в программах на Java представления данных RFML в объекты Record и RecordFormat для последующего использования с другими компонентами IBM Toolbox for Java .

Класс RecordFormatDocument

Класс RecordFormatDocument представляет исходный файл RFML и содержит методы, позволяющие программам на Java выполнять следующие операции:

- Создавать исходные файлы RFML на основе объектов Record, RecordFormat и массивов байт
- Создавать объекты Record, RecordFormat и массивы байт, представляющие информацию, содержащуюся в объекте RecordFormatDocument
- Задавать и получать значения различных объектов и типов данных
- Создавать текст XML (RFML), представляющий данные, содержащиеся в объекте RecordFormatDocument
- Создавать исходный файл RFML, представляемый объектом RecordFormatDocument

For more information about the available methods, see the Javadoc method summary for the RecordFormatDocument class.

Using the RecordFormatDocument class with other IBM Toolbox for Java classes

Класс RecordFormatDocument можно применять вместе со следующими классами IBM Toolbox for Java:

- Классы для работы с записями, в том числе классы доступа на уровне записей (AS400File, SequentialFile и KeyedFile), применяемые для чтения, записи и изменения объектов Record. К этой же категории относится класс LineDataRecordWriter.
- Классы для работы с байтами, включая классы DataQueue, UserSpace и IFSFile, применяющиеся для чтения и записи данных в массив байт.

Не применяйте класс RecordFormatDocument со следующими классами IBM Toolbox for Java, так как эти классы используют операции чтения и записи, не поддерживаемые классом RecordFormatDocument:

- Классы DataArea, так как их методы чтения и записи применимы только к типам данных String, boolean и BigDecimal.
- Классы IFSTextFileInputStream и IFSTextFileOutputStream, так как их методы чтения и записи применимы только к типу данных String.

R • JDBC classes because RFML focuses only on data described by the System i data description specification (DDS).

Информация, связанная с данной

RecordFormatDocument Javadoc

Документы формата записи и синтаксис RFML

Документы RFML, или исходные файлы RFML, содержат теги, определяющие формат данных.

Язык RFML основан на языке PCML, поэтому его синтаксис будет понятен пользователям, знающим PCML. Поскольку RFML является расширением XML, исходные файлы RFML просты для понимания, и их легко создавать. Исходный файл RFML можно создать с помощью обычного текстового редактора. Кроме того, в исходных файлах RFML более четко прослеживается структура данных, чем в исходных файлах различных языков программирования, в том числе Java.

The RFML example Using RFML compared to using IBM Toolbox for Java Record classes includes an example RFML source file.

DTD RFML

Определение типа документа (DTD) RFML задает допустимые элементы и синтаксис RFML. Для того чтобы анализатор XML мог динамически обрабатывать исходные файлы RFML, необходимо объявить DTD RFML в исходном файле:

```
<!DOCTYPE rfml SYSTEM "rfml.dtd">
```

DTD RFML содержится в файле jt400.jar (com/ibm/as400/data/rfml.dtd).

Синтаксис RFML

В DTD RFML определены теги, в свою очередь содержащие теги атрибутов. You use the RFML tags to declare and define the elements in your RFML files.

В следующем примере с помощью синтаксиса RFML описывается один формат записи и одна структура:

```
<rfml>

    <recordformat>
        <data> </data>
    </recordformat>

    <struct>
        <data> </data>
    </struct>

</rfml>
```

Определение типа документа (DTD) RFML:

В этом разделе приведено определение типа документа (DTD) RFML. Данная информация относится к версии 4.0. DTD RFML содержится в файле jt400.jar (com/ibm/as400/data/rfml.dtd).

```
<!--
Определение типа документа Языка описания форматов записей (RFML)

RFML является языком XML. Стандартный формат:
<?xml version="1.0"?>
   <!DOCTYPE rfml SYSTEM "rfml.dtd">
   <rfml version="4.0">
   ...
   </rfml>


(C) Copyright IBM Corporation, 2001,2002
All rights reserved. Licensed Materials Property of IBM
US Government Users Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
-->

<!-- Convenience entities -->
<!ENTITY % string             "CDATA">    <!-- a string of length 0 or greater -->
<!ENTITY % nonNegativeInteger "CDATA">    <!-- a non-negative integer -->
<!ENTITY % binary2            "CDATA">    <!-- an integer in range 0-65535 -->
<!ENTITY % boolean            "(true|false)">
<!ENTITY % datatype  "(char | int | packed | zoned | float | byte | struct)">
<!ENTITY % biditype  "(ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT)">


<!-- The document root element -->
<!ELEMENT rfml (struct | recordformat)+>
<!ATTLIST rfml
          version       %string;    #FIXED "4.0"
```

```
          ccsid         %binary2;             #IMPLIED
>
<!-- Note: The ccsid is the default value that will be used for -->
     <!-- any contained <data type="char"> elements that do not specify a ccsid. -->


<!-- Примечание: RFML не поддерживает вложенные объявления структур.-->
<!-- Все элементы структур должны являться дочерними элементами корневого узла. -->
<!ELEMENT struct (data)+>
<!ATTLIST struct
          name          ID                    #REQUIRED
>


<!-- <!ELEMENT recordformat (data | struct)*> -->
<!ELEMENT recordformat (data)*>
<!ATTLIST recordformat
          name          ID                    #REQUIRED
          description   %string;              #IMPLIED
>
<!-- Примечание: На сервере размер поля text description ограничен 50 байтами. -->

<!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
>
<!-- Примечание: Атрибут name должен быть уникальным в данном формате recordformat. -->
<!-- Примечание: На сервере размер полей Record ограничен 10 байтами. -->
<!-- Примечание: Атрибут length является обязательным, если не задано значение type="struct". -->
<!-- Примечание: Если задано значение type="struct", то атрибут struct является обязательным.-->
<!-- Примечание: Атрибуты ccsid и bidistringtype допустимы, только если задано значение type="char". -->
<!-- Примечание: Атрибут precision можно применять только с типами int, packed и zoned. -->


<!-- The standard predefined character entities -->
<!ENTITY quot  "&#34;">     <!-- quotation mark -->
<!ENTITY amp   "&#38;#38;"> <!-- ampersand -->
<!ENTITY apos  "&#39;">     <!-- apostrophe -->
<!ENTITY lt    "&#38;#60;"> <!-- less than -->
<!ENTITY gt    "&#62;">     <!-- greater than -->
<!ENTITY nbsp  " ">    <!-- non-breaking space -->
<!ENTITY shy   "&#173;">    <!-- soft hyphen (discretionary hyphen) -->
<!ENTITY mdash "&#38;#x2014;">
<!ENTITY ldquo "&#38;#x201C;">
<!ENTITY rdquo "&#38;#x201D;">
```

Тег данных RFML:

The RFML data tag defines a field within a record format or structure.

Listed below are the attributes for the data tag. Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута. Возможные значения атрибута представлены в виде списка, заключенного в фигурные скобки {}, и отделены друг от друга вертикальной чертой |. Для каждого атрибута можно задать только одно значение, которое должно быть указано без скобок.

```
<data type="{ char | int | packed | zoned | float | byte | struct }" ]
    [ bidistringtype="{ ST4 | ST5 | ST6 | ST7 | ST8 | ST9 | ST10 | ST11 | DEFAULT }"]
    [ ccsid="{ число | имя-элемента-данных }" ]
    [ count="{ число | имя-элемента-данных }" ]
    [ init="string" ]
    [ length="{ число | имя-элемента-данных }" ]
    [ name="имя" ]
    [ precision="число" ]
    [ struct="struct-name" ]>
</data>
```

В следующей таблице перечислены атрибуты тега данных. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
type=	*char* Символьное значение. Значение типа *char* возвращается в виде объекта *java.lang.String*. Дополнительная информация приведена в разделе Указание длины с помощью значений *char* . *int* Целое значение. Значение *int* возвращается в виде объекта *java.lang.Long*. Дополнительная информация приведена в разделе Указание длины и точности данных с помощью значений *int*. *packed* Упакованное десятичное значение. Значение *packed* возвращается в виде объекта *java.math.BigDecimal*. Дополнительная информация приведена в разделе Указание длины и точности данных с помощью значений *packed*. *zoned* Зонное десятичное значение. Значение *zoned* возвращается в виде объекта *java.math.BigDecimal*. Дополнительная информация приведена в разделе Указание длины и точности данных с помощью значений *zoned*. *float* Значение с плавающей точкой. Атрибут **length** задает количество байтов: 4 или 8. 4-байтовое целое значение возвращается в виде объекта *java.lang.Float*. В виде объекта *java.lang.Double* возвращается 8-байтовое целое значение. Дополнительная информация приведена в разделе Указание длины данных с помощью значений *float*. *byte* Байтовое значение. Данные не преобразуются. Значение *byte* возвращается в виде массива значений *byte* (*byte[]*). Дополнительная информация приведена в разделе Указание длины данных с помощью значений *byte* . *struct* The name of the **<struct>** element. Объект *struct* позволяет определить структуру и затем несколько раз использовать ее в документе. Конструкция **type=**″struct″ аналогична вставке указанной структуры в документ. Тип *struct* не позволяет задавать длину данных и не поддерживает значение точности.	Задает тип данных (символьный, целочисленный, упакованный, зонный, с плавающей точкой, байтовый или структура). У разных типов данных атрибуты длины и точности принимают разные значения. Дополнительная информация приведена в разделе Значения длины и точности.

Атрибут	Значение	Описание
bidistringtype=	*DEFAULT* where DEFAULT is the default string type for non-bidirectional data (LTR). *ST4* where *ST4* is String Type 4. *ST5* where *ST5* is String Type 5. *ST6* where *ST6* is String Type 6. *ST7* where *ST7* is String Type 7. *ST8* where *ST8* is String Type 8. *ST9* where *ST9* is String Type 9. *ST10* where *ST10* is String Type 10. *ST11* where *ST11* is String Type 11.	Specifies the bidirectional string type for **<data>** elements with **type=**″char″. Если этот атрибут не задан, то тип строки определяется с помощью явно заданного CCSID или CCSID хоста по умолчанию. String types are defined in the Javadoc for the BidiStringType class.
ccsid=	*Число*, где *число* определяет фиксированный неизменяемый CCSID. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя объекта, который будет во время выполнения программы содержать CCSID символьных данных. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=**″int″. See Resolving Relative Names for more information about how relative names are resolved.	Specifies the host coded character set identifier (CCSID) for character data for the **<data>** element. The **ccsid** attribute can be specified only for **<data>** elements with **type=**″char″. Если этот атрибут не задан, то считается, что CCSID символьных данных совпадает с CCSID хоста по умолчанию.
count=	*Число*, где *число* определяет фиксированное неизменяемое число элементов конечного массива. *data-name* where *data-name* defines the name of a **<data>** element within the RFML document that will contain, at runtime, the number of elements in the array. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=**″int″. See Resolving Relative Names for more information about how relative names are resolved.	Указывает, что элемент является массивом указанного размера. Если атрибут count не задан, то элемент не является массивом, хотя он может быть элементом массива.

Атрибут	Значение	Описание
init=	*строка*	Specifies an initial value for the **<data>** element. Начальное значение применяется для инициализации скалярных значений. Если элемент представляет массив или содержится в структуре, определяющей массив, то указанным значением инициализируются все записи массива.
length=	*Число*, где *число* определяет фиксированную неизменяемую длину данных. *data-name* where *data-name* defines the name of a **<data>** element within the RFML document that will contain, at runtime, the length. A *data-name* can be specified only for **<data>** elements with **type=**″char″ or **type=**″byte″. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. In either case, the name must reference a **<data>** element that is defined with **type=**″int″. See Resolving Relative Names for more information about how relative names are resolved.	Задает длину элемента данных. Назначение этого атрибута зависит от типа данных. Дополнительная информация приведена в разделе Значения длины и точности.
name=	*имя*	Specifies the name of the **<data>** element.
precision=	*число*	Задает число значимых разрядов для некоторых числовых типов данных. Дополнительная информация приведена в разделе Значения длины и точности.
struct=	*имя*	Specifies the name of a **<struct>** element for the **<data>** element. A **struct** attribute can be specified only for **<data>** elements with **type=**″struct″.

Информация, связанная с данной

BidiStringType Javadoc

Тег rfml языка RFML:

The rfml tag begins and ends the RFML source file that describes the data format.

Listed below are the attributes for the rfml tag. Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута.

```
<rfml version="версия"
    [ ccsid="number" ]>
</rfml>
```

Список атрибутов тега rfml приведен в следующей таблице. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
version=	*Версия* Фиксированное значение версии DTD RFML . Для версии V5R3 допустимо только значение `4.0`.	Задает версию DTD RFML, применяемую в целях проверки.
ccsid=	*Число* Фиксированное неизменяемое значение идентификатора набора символов (CCSID).	Specifies the host CCSID, which applies to all enclosed **<data type**=″char″> elements that do not specify a CCSID. For more information, see the RFML **<data>** tag. Если этот атрибут не задан, применяется CCSID хоста по умолчанию.

Тег recordformat языка RFML:

The RFML recordformat tag defines a record format, which contains either data elements or references to structure elements.

Listed below are the attributes for the recordformat tag. Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута.

```
<recordformat name="имя"
    [ description="description" ]>
</recordformat>
```

Список атрибутов тега recordformat приведен в следующей таблице. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
name=	*имя*	Задает имя формата записи.
description=	*описание*	Задает описание формата записи.

Тег struct языка RFML:

The RFML struct tag defines a named structure that you can reuse within the RFML source file. Для каждого поля структуры задается тег data.

Listed below are the attributes for the struct tag. Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута.

```
<struct name="name">
</struct>
```

Список атрибутов тега struct приведен в следующей таблице. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
name=	*имя*	Specifies the name of the **<struct>** element.

Анализатор XML и обработчик XSLT

Для применения некоторых пакетов или функций IBM Toolbox for Java необходимо, чтобы в переменной CLASSPATH во время их выполнения был указан анализатор XML или обработчик XSLT.

Ниже приведена информация, которая поможет вам выбрать анализатор и обработчик.

Дополнительная информация о пакетах и функциях IBM Toolbox for Java, для работы которых необходимы анализатор XML и обработчик XSLT приведена на следующей странице:

“Файлы Jar” на стр. 11

Анализатор XML

Если для применения функции или пакета необходим анализатор XML, он должен быть указан в переменной CLASSPATH во время работы программы. Анализатор XML должен соответствовать следующим требованиям:

- Быть совместимым с JAXP
- Расширять класс org.apache.xerces.parsers.SAXParser
- Поддерживать полную проверку схемы

Примечание: Если анализатор необходим только для применения языка XPCML, достаточно, чтобы он поддерживал полную проверку схемы. Если применяется только язык PCML, это требование не является обязательным.

Java 2 Software Developer Kit (J2SDK) версии 1.4 включает анализатор XML, обладающий всеми необходимыми возможностями. Если применяется предыдущая версия J2SDK, для работы с нужным анализатором XML можно:

- Использовать файл x4j400.jar (версия анализатора XML Xerces Apache фирмы IBM)
- Загрузить анализатор XML Xerces с Web-сайта Apache 
R • Use any compatible XML parser in the /QIBM/ProdData/OS400/xml/lib directory on your system

Примечание: Обратите внимание на то, что можно использовать последние версии анализаторов, расположенные в каталоге /QIBM/ProdData/OS400/xml/lib. Например, анализаторы xmlapis11.jar и xerces411.jar позволяют выполнять полную проверку схемы.

Эти анализаторы можно запустить на сервере или скопировать их на рабочую станцию.

Примечание: Любой анализатор XML, позволяющий проверять синтаксис XPCML, поддерживает также проверку PCML и RFML. Обратите внимание на то, что XPCML не поддерживает сохранение данных в двоичном формате.

Обработчик XSLT

Если для применения функции или пакета необходим обработчик XSLT, он должен быть указан в переменной CLASSPATH во время работы программы. Обработчик XSLT должен соответствовать следующим требованиям:

- Быть совместимым с JAXP
- Содержать класс javax.xml.transform.Transformer

Java 2 Software Developer Kit (J2SDK) версии 1.4 включает обработчик XSLT, обладающий всеми необходимыми возможностями. Если применяется предыдущая версия J2SDK, для работы с нужным обработчиком XSLT можно:

- Использовать файл xsltparser.jar (версия обработчика XSLT Xerces Apache фирмы IBM)
- Загрузить процессор XSLT Xalan с Web-сайта Apache 
R • Use any compatible XSLT processor in the /QIBM/ProdData/OS400/xml/lib directory on your system

You can use these processors on your system or copy them to a workstation.

Язык XPCML

Язык XPCML расширяет функциональные возможности и сферу применения PCML благодаря поддержке схем XML. XPCML не поддерживает сохранение в двоичном формате, поэтому, в отличие от PCML, документ XPCML нельзя сохранить в виде потока байтов.

Тем не менее, XPCML обладает некоторыми значительными преимуществами по сравнению с PCML. В частности, он позволяет:

- Задавать и передавать значения параметров программ
- Retrieve the results of a program call to your server in XPCML
- Преобразовывать документы PCML в эквивалентные документы XPCML
- Расширять и настраивать схемы XPCML с помощью новых простых и сложных элементов и атрибутов

Дополнительная информация о XPCML приведена на следующих страницах:

Преимущества XPCML перед PCML
Данный документ содержит дополнительную информацию о расширенных возможностях XPCML по сравнению с PCML.

Требования
Этот документ содержит сведения о программных требованиях для применения XPCML.

Схема и синтаксис XPCML
Этот документ содержит описание определения синтаксиса XPCML и доступных типов данных сервера с помощью схемы XPCML. Он также содержит информацию о том, как можно расширить и настроить схему в соответствии с конкретными требованиями к программной среде

Применение XPCML
В этом разделе описано применение расширенных функций XPCML, таких как передача различных типов данных в качестве параметров программ сервера и получение возвращаемых данных. Он также содержит информацию об уплотнении кода XPCML, которое упрощает его применение и чтение, а также применение XPCML при работе с текущими приложениями, поддерживающими PCML.

XPCML - это единственный инструмент, позволяющий применять XML на сервере. Дополнительная информация о применении XML приведена на следующих страницах:

Компоненты XML

XML Toolkit

Домен архитектуры W3C: Схема XML 

Преимущества XPCML по сравнению с PCML

Язык XPCML обладает несколькими значительными преимуществами по сравнению с PCML.

- Указание и передача значений параметров программ
- R Retrieve the results of a program call to your System i5 in XPCML
- Позволяет преобразовывать документы PCML в эквивалентные документы XPCML
- Позволяет расширять и настраивать схемы XPCML с помощью новых простых и сложных элементов и атрибутов

Указание и передача значений параметров программ

В XPCML типы параметров программ определяются с помощью схемы XML; в PCML применяются определения типов данных (DTD). Во время анализа синтаксиса анализатор XML сравнивает переданные значения параметров с соответствующими определениями схемы. Схема должна содержать несколько типов данных: строчные, целые, длинные целые значения и т.д. Возможность указания и передачи значений параметров программ является значительным преимуществом по сравнению с PCML. В PCML можно проверять значения параметров только после анализа синтаксиса документа PCML. Кроме того, для

проверки значений параметров в PCML часто необходимо создавать отдельное приложение.

Получение результатов работы программ в формате XPCML

XPCML также позволяет получать результаты работы программ в формате XPCML. В PCML для получения результатов вызова программы необходимо после ее завершения применить один из методов getValue класса ProgramCallDocument. XPCML позволяет как воспользоваться методами getValue, так и вызвать в программе XPCML метод generateXPCML, который возвращает результаты вызова программы в виде документа XPCML.

Преобразование документов PCML в XPCML

R С помощью нового метода класса ProgramCallDocument, transformPCMLToXPCML, можно преобразовывать
R существующие документы PCML в эквивалентные документы XPCML. This allows you to take advantage of new
R XPCML function without writing XPCML source for your existing System i5 program call documents.

Расширение и настройка схемы XPCML

XPCML является расширяемым языком, что позволяет определять новые типы параметров схемы XPCML. При уплотнении XPCML схема XPCML расширяется за счет новых определений типов данных, которые упрощают и расширяют возможности чтения и применения документов XPCML.

Требования для применения XPCML

The Extensible Program Call Markup Language (XPCML) has the same workstation Java virtual machine requirements as the rest of the IBM Toolbox for Java.

Дополнительная информация приведена на следующей странице:

"Требования к рабочей станции для запуска приложений IBM Toolbox for Java" на стр. 8

Кроме того, при работе с XPCML должны быть выполнены следующие требования к файлу схемы XML, анализатору XML и обработчику XSLT:

Файл схемы XML

Документы XPCML должны содержать информацию о расположении файла, содержащего схему. По умолчанию в IBM Toolbox for Java применяется файл схемы xpcml.xsd, расположенный в файле jt400.jar.

Для использования схемы по умолчанию извлеките файл xpcml.xsd из файла jt400.jar и поместите в нужный каталог. Ниже описан один из способов извлечения файла .xsd на рабочей станции.

Извлечение файла схемы xpcml.xsd

- Запустите сеанс командной строки в каталоге, в котором находится файл jt400.jar
- Извлеките файл .xsd с помощью следующей команды:

```
jar xvf  jt400.jar   com/ibm/as400/data/xpcml.xsd
```

Примечание: Для того чтобы не запускать команду в каталоге, содержащем файл jt400.jar, можно указать полный путь к файлу jt400.jar.

Файл схемы по умолчанию (или любой файл схемы) можно поместить в любой каталог. The only requirement is that you specify the location of the schema file by using the xsi:noNamespaceSchemaLocation attribute in the <xpcml> tag.

Расположение схемы можно указать в виде полного пути файла или в виде URL.

Примечание: Несмотря на то, что в следующих примерах применяется файл схемы xpcml.xsd, можно указать любую схему, которая является расширением xpcml.xsd.

- Для того чтобы использовать схему, расположенную в каталоге файла XPCML, введите команду xsi:noNamespaceSchemaLocation='xpcml.xsd'
- Для того чтобы задать полный путь к файлу схемы, введите следующую команду: xsi:noNamespaceSchemaLocation='c:\myDir\xpcml.xsd'
- Для того чтобы задать URL файла схемы, введите следующую команду: xsi:noNamespaceSchemaLocation='http://myServer/xpcml.xsd'

Для того чтобы определить версию HTML файла xpcml.xsd file, откройте следующую страницу:

"Файл схемы xpcml.xsd" на стр. 424

Анализатор XML и обработчик XSLT

Во время выполнения программы анализатор XML и обработчик XSLT должны быть указаны в переменной среды CLASSPATH. Дополнительная информация приведена на следующей странице:

"Анализатор XML и обработчик XSLT" на стр. 418

Схема и синтаксис XPCML

XPCML documents, called XPCML source files, contain tags and data that fully define calls to programs on your system.

Поскольку в XPCML вместо определений типов данных (DTD) применяются схемы XML, язык XPCML позволяет выполнять операции, не поддерживаемые PCML:

- Передавать значения входных параметров программ в виде элементов XML
- Получать значения выходных параметров в виде элементов XML
- Автоматически проверять значения, передаваемые программам, с помощью анализатора XML
- Расширять схему с помощью определений новых простых и сложных элементов

Дополнительная информация о схеме и синтаксисе XPCML приведена на следующих страницах:

Сравнение исходных файлов XPCML и PCML
Ознакомьтесь с примерами сравнения исходных файлов XPCML и PCML. Примеры демонстрируют расширенные возможности XPCML и простоту и легкость чтения исходных файлов в этом формате.

Схема XPCML
Просмотрите файл схемы XPCML, который содержит дополнительную информацию о применении и расширении схемы XPCML.

Синтаксис XPCML
Просмотрите список элементов синтаксиса XPCML, с помощью которых в схеме определяются элементы XPCML.

Атрибуты тегов XPCML
Этот раздел содержит описание различных атрибутов каждого элемента, определенного в схеме XPCML.

Сравнение исходных файлов XPCML и PCML:

XPCML отличается от PCML по нескольким показателям, но основная разница заключается в том, что XPCML позволяет задавать значения входных параметров в исходном файле XPCML.

PCML allows you to use the init attribute of the <data> tag to specify the initial value for a data element in the PCML source. Тем не менее, при указании значений PCML действуют следующие ограничения:

- С помощью атрибута init нельзя задавать массивы значений
- Проверка значения init выполняется только после анализа документа PCML

Для того чтобы задать с помощью PCML массив значений, необходимо считать и проанализировать документ PCML, после чего многократно вызвать метод ProgramCallDocument.setValue().

XPCML упрощает указание значений отдельных элементов и массивов:

- Указание значений отдельных элементов и массивов элементов в исходном файле XPCML
- Проверка заданных значений массивов при анализе документа

R Ниже приведены несколько простых сравнений, иллюстрирующих отличия XPCML от PCML. Each example
R defines a program call for a System i program.

Example: Calling a System i program

R The following examples call a System i program called prog1.

Исходный код XPCML

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation='xpcml.xsd' version="4.0">
    <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <parameterList>
         <stringParm name="parm1"  passDirection="in" length="10">Parm1</stringParm>
         <intParm    name="parm2"  passDirection="in">5</intParm>
         <shortParm  name="parm3"  passDirection="in">3</shortParm>
      </parameterList>
    </program>
</xpcml>
```

Исходный код PCML

```
<pcml version="4.0">
  <program name="prog1" path="QSYS.LIB/MYLIB.LIB/PROG1.PGM">
     <data name="parm1"  type="char" usage="input" length="10" init="Parm1"/>
     <data name="parm2"  type="int"  usage="input" length="4"  init="5"/>
     <data name="parm3"  type="int"  usage="input" length="2"  precision="16" init="3"/>
  </program>
</pcml>
```

Example: Calling a System i program using an array of string parameters

R The following examples call a System i program called prog2 and define parm1 as an array of string parameters.
R Обратите внимание на следующие функциональные возможности языка XPCML:

- Инициализация значения каждого элемента в массиве
- Указание входных значений в виде содержимого элементов, которое может проверить анализатор XML, поддерживающий полную проверку схем

Воспользоваться этими функциональными возможностями XPCML можно без создания специального кода на Java.

PCML обладает значительно меньшей производительностью, чем XPCML. PCML не позволяет инициализировать значение каждого элемента в массиве. PCML также не поддерживает проверку значений инициализации во время анализа документа. Для выполнения операций, поддерживаемых XPCML, необходимо считать и выполнить анализ документа PCML и создать приложение на Java, которое будет задавать значение каждого элемента массива. Необходимо также создать программу для проверки значений параметров.

Исходный код XPCML

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

    <program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG2.PGM">
      <parameterList>
         <arrayOfStringParm name="parm1" passDirection="in"
            length="10" count="3">
             <i>Parm1-First value</i>
             <i>Parm1-Second value</i>
             <i>Parm1-Third value</i>
         </arrayOfStringParm>
         <longParm name="parm2"  passDirection="in">5</longParm>
         <zonedDecimalParm name="parm3"  passDirection="in"
            totalDigits="5" fractionDigits="2">32.56</zonedDecimalParm>
      </parameterList>
    </program>
</xpcml>
```

Исходный код PCML

```
<pcml version="4.0">
  <program name="prog2" path="QSYS.LIB/MYLIB.LIB/PROG2.PGM">
     <data name="parm1"  type="char"  usage="input" length="20" count="3"/>
     <data name="parm2"  type="int"   usage="input" length="8"  init="5"/>
     <data name="parm3"  type="zoned" usage="input" length="5"  precision="2" init="32.56"/>
  </program>
</pcml>
```

Файл схемы xpcml.xsd:

Для упрощения просмотра и печати документа некоторые строки этой версии HTML файла xpcml.xsd перенесены. Эти же строки в исходном файле .xsd не переносятся.

Дополнительная информация о применении файла xpcml.xsd приведена в разделе Требования для применения XPCML.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
R <?xml version="1.0" encoding="UTF-8"?>
R
R <!--////////////////////////////////////////////////////////////////////////
R //
R // JTOpen (IBM Toolbox for Java - версия OSS)
R //
R // Имя файла: xpcml.xsd
R //
R // На исходный код в данном примере распространяется действие Общей лицензии IBM
R // версии 1.0, утвержденной компанией Open Source Initiative.
R // Copyright (C) 1997-2008 International Business Machines Corporation and
R // others. Все права защищены.
R //
R ////////////////////////////////////////////////////////////////////////////////-->
R
R <xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
R
R <xs:annotation>
R  <xs:documentation>
R   Схема для документов xpcml (расширенного языка вызова программ).
R  </xs:documentation>
R </xs:annotation>
R
R <xs:element name="xpcml">
R   <xs:complexType>
R     <xs:sequence>
```

```
R       <xs:element ref="structOrProgram" minOccurs="1" maxOccurs="unbounded" />
R     </xs:sequence>
R     <xs:attribute name="version" use="required">
R      <xs:simpleType>
R        <xs:restriction base="xs:string">
R          <xs:enumeration value="4.0"/>
R        </xs:restriction>
R      </xs:simpleType>
R    </xs:attribute>
R   </xs:complexType>
R
R   <!-- Define key/keyref link between the name of a struct   -->
R   <!-- атрибутом struct поля параметра.        -->
R   <xs:key name="StructKey">
R     <xs:selector xpath="struct"/>
R     <xs:field xpath="@name"/>
R   </xs:key>
R   <xs:keyref name="spRef" refer="StructKey">
R      <xs:selector xpath="structParm" />
R      <xs:field xpath="@struct" />
R   </xs:keyref>
R </xs:element>
R
R <!-- Program tag and attributes -->
R <xs:element name="program" substitutionGroup="structOrProgram">
R   <xs:complexType>
R     <xs:sequence>
R       <xs:element ref="parameterList" minOccurs="1" maxOccurs="1"/>
R          <!-- Применяется в качестве тега-заменителя для списка параметров программы. -->
R     </xs:sequence>
R     <!-- Имя вызываемой программы. -->
R     <xs:attribute name="name" type="string50" use="required" />
R     <!-- Путь к объекту программы. По умолчанию она расположена в библиотеке QSYS. -->
R     <xs:attribute name="path" type="xs:string"/>
R     <!-- Указание порядка анализа параметров. -->
R     <!-- Значение - это список имен параметров, разделенных пробелами. -->
R     <xs:attribute name="parseOrder" type="xs:string"/>
R     <!-- Имя точки входа служебной программы. -->
R     <xs:attribute name="entryPoint" type="xs:string"/>
R     <!-- Тип значения, если оно возвращается служебной программой. -->
R     <xs:attribute name="returnValue" type="returnValueType"/>
R     <!-- When calling a Java program andSystem i -->
R     <!-- program is on same server -->
R     <!-- and is thread-safe, set to true to call the
R     <!-- System i program in same job -->
R     <!-- программа iSeries вызывалась в том же задании и в той же нити, что и программа на Java. -->
R     <xs:attribute name="threadSafe" type="xs:boolean"  />
R     <!-- CCSID имени точки входа в служебной программе. -->
R     <xs:attribute name="epccsid" type="ccsidType"/>
R   </xs:complexType>
R </xs:element>
R
R <!-- Список состоит из одного или нескольких параметров. -->
R <xs:element name="parameterList">
R   <xs:complexType>
R       <xs:group ref="programParameter" minOccurs="1" maxOccurs="unbounded"/>
R    </xs:complexType>
R </xs:element>
R
R <!-- Все поддерживаемые типы параметров программы. -->
R <xs:group name="programParameter">
R       <xs:choice>
R  <xs:element ref="stringParmGroup"/>
R  <xs:element ref="stringParmArrayGroup"/>
R  <xs:element ref="intParmGroup"/>
R  <xs:element ref="intParmArrayGroup"/>
R         <xs:element ref="unsignedIntParmGroup"/>
```

```
R         <xs:element ref="unsignedIntParmArrayGroup"/>
R  <xs:element ref="shortParmGroup"/>
R  <xs:element ref="shortParmArrayGroup"/>
R  <xs:element ref="unsignedShortParmGroup"/>
R  <xs:element ref="unsignedShortParmArrayGroup"/>
R  <xs:element ref="longParmGroup"/>
R  <xs:element ref="longParmArrayGroup"/>
R  <xs:element ref="zonedDecimalParmGroup"/>
R  <xs:element ref="zonedDecimalParmArrayGroup"/>
R  <xs:element ref="packedDecimalParmGroup"/>
R         <xs:element ref="packedDecimalParmArrayGroup"/>
R  <xs:element ref="floatParmGroup"/>
R  <xs:element ref="floatParmArrayGroup"/>
R  <xs:element ref="doubleParmGroup"/>
R  <xs:element ref="doubleParmArrayGroup"/>
R  <xs:element ref="hexBinaryParmGroup"/>
R  <xs:element ref="hexBinaryParmArrayGroup"/>
R         <xs:element ref="structParmGroup"/>
R         <xs:element ref="structParmArrayGroup"/>
R         <xs:element ref="structArrayGroup"/>
R         <xs:element ref="struct"/>
R       </xs:choice>
R
R </xs:group>
R
R <!-- Абстрактный тип для всех типов параметров. -->
R <xs:element name="stringParmGroup" type="stringParmType" abstract="true" />
R <xs:element name="stringParmArrayGroup" type="stringParmArrayType" abstract="true" />
R <xs:element name="intParmGroup" type="intParmType" abstract="true" />
R <xs:element name="intParmArrayGroup" type="intParmArrayType" abstract="true" />
R <xs:element name="unsignedIntParmGroup" type="unsignedIntParmType" abstract="true" />
R <xs:element name="unsignedIntParmArrayGroup" type="unsignedIntParmArrayType" abstract="true" />
R <xs:element name="shortParmGroup" type="shortParmType" abstract="true" />
R <xs:element name="shortParmArrayGroup" type="shortParmArrayType" abstract="true" />
R <xs:element name="unsignedShortParmGroup" type="unsignedShortParmType" abstract="true" />
R <xs:element name="unsignedShortParmArrayGroup" type="unsignedShortParmArrayType" abstract="true" />
R <xs:element name="longParmGroup" type="longParmType" abstract="true" />
R <xs:element name="longParmArrayGroup" type="longParmArrayType" abstract="true" />
R <xs:element name="zonedDecimalParmGroup" type="zonedDecimalParmType" abstract="true" />
R <xs:element name="zonedDecimalParmArrayGroup" type="zonedDecimalParmArrayType" abstract="true" />
R <xs:element name="packedDecimalParmGroup" type="packedDecimalParmType" abstract="true" />
R <xs:element name="packedDecimalParmArrayGroup" type="packedDecimalParmArrayType" abstract="true" />
R <xs:element name="floatParmGroup" type="floatParmType" abstract="true" />
R <xs:element name="floatParmArrayGroup" type="floatParmArrayType" abstract="true" />
R <xs:element name="doubleParmGroup" type="doubleParmType" abstract="true" />
R <xs:element name="doubleParmArrayGroup" type="doubleParmArrayType" abstract="true" />
R <xs:element name="hexBinaryParmGroup" type="hexBinaryParmType" abstract="true" />
R <xs:element name="hexBinaryParmArrayGroup" type="hexBinaryParmArrayType" abstract="true" />
R <xs:element name="structParmGroup" type="structParmType" abstract="true" />
R <xs:element name="structParmArrayGroup" type="structParmArrayType" abstract="true" />
R <xs:element name="structArrayGroup" type="structArrayType" abstract="true"
R             substitutionGroup="structOrProgram" />
R
R <!-- String parameter -->
R    <xs:element name="stringParm" type="stringParmType" substitutionGroup="stringParmGroup"
R                nillable="true"/>
R      <xs:complexType name="stringParmType">
R        <xs:simpleContent>
R           <xs:extension base="stringFieldType">
R             <xs:attributeGroup ref="commonParmAttrs"/>
R           </xs:extension>
R        </xs:simpleContent>
R    </xs:complexType>
R
R <!-- Array of string parameters -->
R   <xs:element name="arrayOfStringParm" type="stringParmArrayType"
R               substitutionGroup="stringParmArrayGroup" nillable="true" />
```

```
R   <xs:complexType name="stringParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="stringElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R     <!-- Число элементов массива. -->
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <!-- Число символов в каждом параметре string. -->
R     <xs:attribute name="length" type="xs:string"/>
R     <!-- CCSID хоста для каждого параметра string. -->
R     <xs:attribute name="ccsid" type="xs:string"/>
R     <!-- Задает способ усечения пробелов (слева, справа, слева и справа, не усекать). -->
R     <xs:attribute name="trim" type="trimType" />
R     <!-- Размер каждого символа (chartype в PCML). -->
R     <xs:attribute name="bytesPerChar" type="charType" />
R    <!-- Двунаправленные строки (string). -->
R     <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
R   </xs:complexType>
R
R       <xs:complexType name="stringElementType">
R         <xs:simpleContent>
R            <xs:extension base="xs:string">
R                 <!-- Указатель на элемент массива. -->
R                 <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R            </xs:extension>
R         </xs:simpleContent>
R        </xs:complexType>
R
R
R <!-- Integer parameter (4 bytes on server) -->
R     <xs:element name="intParm" type="intParmType"  nillable="true" substitutionGroup="intParmGroup"   />
R       <xs:complexType name="intParmType" >
R          <xs:simpleContent>
R             <xs:extension base="intFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R          </xs:simpleContent>
R       </xs:complexType>
R
R <!-- intParm array type -->
R   <xs:element name="arrayOfIntParm" type="intParmArrayType" substitutionGroup="intParmArrayGroup"
R               nillable="true" />
R   <xs:complexType name="intParmArrayType">
R     <xs:sequence>
R       <!-- i - это тег для указания элементов массива, не являющихся структурами. -->
R       <xs:element name="i" type="intElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="intElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:int">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R
R <!-- Unsigned Integer parameter (4 bytes on server) -->
R     <xs:element name="unsignedIntParm" type="unsignedIntParmType" nillable="true"
R                 substitutionGroup="unsignedIntParmGroup"   />
```

```
R       <xs:complexType name="unsignedIntParmType">
R          <xs:simpleContent>
R             <xs:extension base="unsignedIntFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R          </xs:simpleContent>
R       </xs:complexType>
R
R <!-- unsigned intParm array type -->
R   <xs:element name="arrayOfUnsignedIntParm" type="unsignedIntParmArrayType"
R               substitutionGroup="unsignedIntParmArrayGroup" nillable="true" />
R   <xs:complexType name="unsignedIntParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="unsignedIntElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="unsignedIntElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:unsignedInt">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R
R <!-- Short integer parameter (2 bytes on server) -->
R    <xs:element name="shortParm" type="shortParmType" nillable="true" substitutionGroup="shortParmGroup"/>
R       <xs:complexType name="shortParmType">
R          <xs:simpleContent>
R             <xs:extension base="shortFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R          </xs:simpleContent>
R       </xs:complexType>
R
R <!-- shortParm array type -->
R   <xs:element name="arrayOfShortParm" type="shortParmArrayType" substitutionGroup="shortParmArrayGroup"
R               nillable="true" />
R   <xs:complexType name="shortParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="shortElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="shortElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:short">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R
R <!-- Unsigned Short integer parameter (2 bytes on server) -->
R     <xs:element name="unsignedShortParm" type="unsignedShortParmType" nillable="true"
R                 substitutionGroup="unsignedShortParmGroup"   />
R       <xs:complexType name="unsignedShortParmType">
R          <xs:simpleContent>
```

```
R             <xs:extension base="unsignedShortFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R          </xs:simpleContent>
R       </xs:complexType>
R
R <!-- unsignedShortParm array type -->
R   <xs:element name="arrayOfUnsignedShortParm" type="unsignedShortParmArrayType"
R               substitutionGroup="unsignedShortParmArrayGroup" nillable="true" />
R   <xs:complexType name="unsignedShortParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="unsignedShortElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="unsignedShortElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:unsignedShort">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R
R <!-- Long integer parameter (8 bytes on server) -->
R     <xs:element name="longParm" type="longParmType" nillable="true" substitutionGroup="longParmGroup"  />
R       <xs:complexType name="longParmType">
R          <xs:simpleContent>
R             <xs:extension base="longFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R          </xs:simpleContent>
R       </xs:complexType>
R
R <!-- longParm array type -->
R   <xs:element name="arrayOfLongParm" type="longParmArrayType" substitutionGroup="longParmArrayGroup"
R               nillable="true" />
R   <xs:complexType name="longParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="longElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="longElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:long">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R
R <!-- ZonedDecimal parameter -->
R     <xs:element name="zonedDecimalParm" type="zonedDecimalParmType" nillable="true"
R                 substitutionGroup="zonedDecimalParmGroup"   />
R       <xs:complexType name="zonedDecimalParmType">
R          <xs:simpleContent>
R             <xs:extension base="zonedDecimalFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
```

```
R             </xs:extension>
R           </xs:simpleContent>
R       </xs:complexType>
R
R <!-- zonedDecimalParm array type -->
R   <xs:element name="arrayOfZonedDecimalParm" type="zonedDecimalParmArrayType"
R               substitutionGroup="zonedDecimalParmArrayGroup" nillable="true" />
R   <xs:complexType name="zonedDecimalParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="zonedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <!-- Общее число разрядов в поле (атрибут length в PCML). -->
R     <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
R     <!-- Число дробных разрядов (атрибут precision в PCML). -->
R     <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
R   </xs:complexType>
R
R   <xs:complexType name="zonedDecimalElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:decimal">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R <!-- packedDecimal parameter -->
R     <xs:element name="packedDecimalParm" type="packedDecimalParmType" nillable="true"
R                 substitutionGroup="packedDecimalParmGroup"   />
R       <xs:complexType name="packedDecimalParmType">
R          <xs:simpleContent>
R             <xs:extension base="packedDecimalFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R          </xs:simpleContent>
R       </xs:complexType>
R
R <!-- packedDecimalParm array type -->
R   <xs:element name="arrayOfPackedDecimalParm" type="packedDecimalParmArrayType"
R               substitutionGroup="packedDecimalParmArrayGroup" nillable="true" />
R   <xs:complexType name="packedDecimalParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="packedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
R     <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
R   </xs:complexType>
R
R   <xs:complexType name="packedDecimalElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:decimal">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R <!-- Float parameter (4 bytes on server) -->
R    <xs:element name="floatParm" type="floatParmType" nillable="true" substitutionGroup="floatParmGroup"/>
R       <xs:complexType name="floatParmType">
R          <xs:simpleContent>
```

```
R             <xs:extension base="floatFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R           </xs:simpleContent>
R       </xs:complexType>
R
R <!-- floatParm array type -->
R   <xs:element name="arrayOfFloatParm" type="floatParmArrayType" substitutionGroup="floatParmArrayGroup"
R               nillable="true" />
R   <xs:complexType name="floatParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="floatElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="floatElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:float">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R
R <!-- Double parameter (8 bytes on server) -->
R     <xs:element name="doubleParm" type="doubleParmType" nillable="true"
R                 substitutionGroup="doubleParmGroup"   />
R       <xs:complexType name="doubleParmType">
R          <xs:simpleContent>
R             <xs:extension base="doubleFieldType">
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R          </xs:simpleContent>
R       </xs:complexType>
R
R <!-- doubleParm array type -->
R   <xs:element name="arrayOfDoubleParm" type="doubleParmArrayType"
R               substitutionGroup="doubleParmArrayGroup" nillable="true" />
R   <xs:complexType name="doubleParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="doubleElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="doubleElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:double">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R
R <!-- Hex binary parameter (any number of bytes; unsigned) -->
R     <xs:element name="hexBinaryParm" type="hexBinaryParmType"  substitutionGroup="hexBinaryParmGroup"  />
R       <xs:complexType name="hexBinaryParmType">
R          <xs:simpleContent>
R             <xs:extension base="hexBinaryFieldType">
R               <!-- Длина поля в байтах (атрибут length в PCML). -->
```

```
R               <xs:attribute name="totalBytes" type="xs:string"/>
R               <xs:attributeGroup ref="commonParmAttrs"/>
R             </xs:extension>
R           </xs:simpleContent>
R       </xs:complexType>
R
R <!-- hexBinaryParm array type -->
R   <xs:element name="arrayOfHexBinaryParm" type="hexBinaryParmArrayType"
R               substitutionGroup="hexBinaryParmArrayGroup" nillable="true" />
R   <xs:complexType name="hexBinaryParmArrayType">
R     <xs:sequence>
R       <xs:element name="i" type="hexBinaryElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <xs:attribute name="totalBytes" type="xs:string"/>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R   </xs:complexType>
R
R   <xs:complexType name="hexBinaryElementType">
R     <xs:simpleContent>
R        <xs:extension base="xs:hexBinary">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:simpleContent>
R   </xs:complexType>
R
R <!-- Structure parm type -->
R   <xs:element name="structParm" type="structParmType"  substitutionGroup="structParmGroup"   />
R       <xs:complexType name="structParmType">
R          <xs:complexContent>
R             <xs:extension base="structureParmArray">
R               <xs:attribute name="struct" type="string50"/>
R               <!-- Specifies whether the parameter is passed by value or reference ('passby' in PCML).-->
R               <!-- Значение допустимо только для целочисленных типов. -->
R               <xs:attribute name="passMode" type="passModeType"/>
R               <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R               <xs:attribute name="count" type="xs:string"/>
R             </xs:extension>
R           </xs:complexContent>
R       </xs:complexType>
R
R <!-- Structure parm array type -->
R   <xs:element name="arrayOfStructParm" type="structParmArrayType"
R               substitutionGroup="structParmArrayGroup" nillable="true" />
R   <xs:complexType name="structParmArrayType">
R     <xs:sequence>
R         <!-- тег struct_i представляет отдельные параметры или элементы массива параметров struct. -->
R         <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <xs:attributeGroup ref="commonParmAttrs"/>
R     <xs:attributeGroup ref="commonFieldAttrs"/>
R     <xs:attribute name="struct" type="string50"/>
R   </xs:complexType>
R
R   <xs:complexType name="structElementType">
R     <xs:complexContent>
R        <xs:extension base="structureParmArray">
R             <xs:attribute name="index" type="xs:nonNegativeInteger"  />
R        </xs:extension>
R     </xs:complexContent>
R   </xs:complexType>
R
R
```

```
R <!-- Struct element -->
R   <xs:element name="struct" type="structureParmArray" substitutionGroup="structOrProgram" />
R
R <!-- Struct array type -->
R   <xs:element name="arrayOfStruct" type="structArrayType" substitutionGroup="structArrayGroup"
R               nillable="true" />
R   <xs:complexType name="structArrayType">
R     <xs:sequence>
R        <!-- тег struct_i tag представляет элементы struct в массиве. -->
R        <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
R     </xs:sequence>
R     <!-- Имя элемента struct. -->
R     <xs:attribute name="name" type="string50"/>
R     <!-- Число элементов массива. -->
R     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R     <xs:attribute name="count" type="xs:string" />
R     <!-- Указывает, для чего предназначен данный элемент struct:  -->
R     <!-- для ввода, для вывода или для ввода-вывода (атрибут usage в PCML). -->
R     <xs:attribute name="passDirection" type="passDirectionType"/>
R     <!-- Смещение элемента struct в выходном параметре. -->
R     <xs:attribute name="offset" type="xs:string" />
R     <!-- Базовое расположение для атрибута offset. -->
R     <xs:attribute name="offsetFrom" type="xs:string" />
R     <!-- Число байтов, которые следует резервировать для выходных данных элемента. -->
R     <xs:attribute name="outputSize" type="xs:string" />
R     <!-- Самая ранняя версия системы i5/OS, поддерживающая данный элемент. -->
R     <xs:attribute name="minvrm" type="string10" />
R     <!-- Самая поздняя версия системы i5/OS, поддерживающая данный элемент. -->
R     <xs:attribute name="maxvrm" type="string10" />
R   </xs:complexType>
R
R
R <!-- Общие атрибуты для всех типов полей. -->
R   <xs:attributeGroup name="commonParmAttrs">
R      <!-- Указывает, для чего предназначен данный параметр:  -->
R      <!-- для ввода, для вывода или для ввода-вывода (атрибут usage в PCML). -->
R      <!-- Если значение не задано, по умолчанию применяется значение inherit. -->
R      <xs:attribute name="passDirection" type="passDirectionType"/>
R      <!-- Указывает, передается ли значение параметра или ссылка на него (атрибут passby в PCML).--> -->
R      <!-- Если значение не задано, по умолчанию применяется значение reference. -->
R      <xs:attribute name="passMode" type="passModeType" />
R      <!-- Смещение элемента в выходном параметре. -->
R      <!-- The default value if none is specified is 0. -->
R      <xs:attribute name="offset" type="xs:string" />
R      <!-- Базовое расположение для атрибута offset. -->
R      <xs:attribute name="offsetFrom" type="xs:string" />
R      <!-- Число байтов, которые следует резервировать для выходных данных элемента. -->
R      <xs:attribute name="outputSize" type="xs:string" />
R      <!-- Самая ранняя версия системы i5/OS, поддерживающая данное поле. -->
R      <!-- Если значение не задано, считается, что поле поддерживается всеми версиями. -->
R      <xs:attribute name="minvrm" type="string10" />
R      <!-- Самая поздняя версия системы i5/OS, поддерживающая данное поле. -->
R      <!-- Если значение не задано, считается, что поле поддерживается всеми версиями. -->
R      <xs:attribute name="maxvrm" type="string10" />
R   </xs:attributeGroup>
R
R   <xs:simpleType name="passDirectionType">
R     <xs:restriction base="xs:string">
R        <xs:enumeration value="in"/>
R        <xs:enumeration value="inout"/>
R        <xs:enumeration value="out"/>
R        <xs:enumeration value="inherit"/>
R     </xs:restriction>
R   </xs:simpleType>
R
R   <xs:simpleType name="passModeType">
R     <xs:restriction base="xs:string">
```

```
R         <xs:enumeration value="value"/>
R         <xs:enumeration value="reference"/>
R       </xs:restriction>
R    </xs:simpleType>
R
R    <!-- Following types are to maintain compatibility with PCML -->
R    <xs:simpleType name="bidiStringTypeType">
R      <xs:restriction base="xs:string">
R         <xs:enumeration value="ST4"/>
R         <xs:enumeration value="ST5"/>
R         <xs:enumeration value="ST6"/>
R         <xs:enumeration value="ST7"/>
R         <xs:enumeration value="ST8"/>
R         <xs:enumeration value="ST9"/>
R         <xs:enumeration value="ST10"/>
R         <xs:enumeration value="ST11"/>
R         <xs:enumeration value="DEFAULT"/>
R      </xs:restriction>
R    </xs:simpleType>
R
R    <xs:simpleType name="charType">
R      <xs:restriction base="xs:string">
R         <xs:enumeration value="onebyte"/>
R         <xs:enumeration value="twobyte"/>
R       </xs:restriction>
R    </xs:simpleType>
R
R    <xs:simpleType name="trimType">
R      <xs:restriction base="xs:string">
R         <xs:enumeration value="none"/>
R         <xs:enumeration value="left"/>
R         <xs:enumeration value="right"/>
R         <xs:enumeration value="both"/>
R       </xs:restriction>
R    </xs:simpleType>
R
R    <xs:simpleType name="returnValueType">
R      <xs:restriction base="xs:string">
R         <xs:enumeration value="void"/>
R         <xs:enumeration value="integer"/>
R       </xs:restriction>
R    </xs:simpleType>
R
R
R    <xs:complexType name="structureParmArray">
R      <xs:sequence>
R         <xs:group ref="structureParm" minOccurs="0" maxOccurs="unbounded"/>
R      </xs:sequence>
R      <xs:attribute name="name" type="string50"/>
R      <xs:attribute name="passDirection" type="passDirectionType"/>
R      <xs:attribute name="offset" type="xs:string" />
R      <xs:attribute name="offsetFrom" type="xs:string" />
R      <xs:attribute name="outputSize" type="xs:string" />
R      <xs:attribute name="minvrm" type="string10" />
R      <xs:attribute name="maxvrm" type="string10" />
R    </xs:complexType>
R
R
R <!-- Параметр structureParm имеет один из следующих типов: stringParm, intParm,
R      shortParm, longParm, zonedDecimalParm, packedDecimalParm, floatParm,
R      doubleParm или hexBinaryParm. -->
R       <xs:group name="structureParm">
R         <xs:choice>
R           <xs:element ref="stringParmGroup"  />
R           <xs:element ref="stringParmArrayGroup"  />
R           <xs:element ref="intParmGroup"   />
R           <xs:element ref="intParmArrayGroup"   />
```

```
R           <xs:element ref="unsignedIntParmGroup"   />
R           <xs:element ref="unsignedIntParmArrayGroup"   />
R           <xs:element ref="shortParmGroup" />
R           <xs:element ref="shortParmArrayGroup" />
R           <xs:element ref="unsignedShortParmGroup"   />
R           <xs:element ref="unsignedShortParmArrayGroup"   />
R           <xs:element ref="longParmGroup"  />
R           <xs:element ref="longParmArrayGroup"  />
R           <xs:element ref="zonedDecimalParmGroup"  />
R           <xs:element ref="zonedDecimalParmArrayGroup"  />
R           <xs:element ref="packedDecimalParmGroup" />
R           <xs:element ref="packedDecimalParmArrayGroup" />
R           <xs:element ref="floatParmGroup" />
R           <xs:element ref="floatParmArrayGroup" />
R           <xs:element ref="doubleParmGroup" />
R           <xs:element ref="doubleParmArrayGroup" />
R           <xs:element ref="hexBinaryParmGroup" />
R           <xs:element ref="hexBinaryParmArrayGroup" />
R           <xs:element ref="structParmGroup" />
R           <xs:element ref="structParmArrayGroup"/>
R           <xs:element ref="structArrayGroup"/>
R           <xs:element ref="struct"/>
R         </xs:choice>
R       </xs:group>
R
R
R <!-- Схема определений полей. -->
R
R   <!-- Define basic System i native data types. -->
R
R      <xs:complexType name="zonedDecimal">
R         <xs:simpleContent>
R            <xs:extension base="xs:decimal">
R                 <xs:attribute name="totalDigits" type="xs:positiveInteger"  />
R                 <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger" />
R            </xs:extension>
R         </xs:simpleContent>
R      </xs:complexType>
R
R
R      <xs:complexType name="packedDecimal">
R         <xs:simpleContent>
R            <xs:extension base="xs:decimal">
R                 <xs:attribute name="totalDigits" type="xs:positiveInteger" />
R                 <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger" />
R            </xs:extension>
R         </xs:simpleContent>
R      </xs:complexType>
R
R
R     <xs:complexType name="structureFieldArray">
R      <xs:sequence>
R         <xs:group ref="structureField" minOccurs="1" maxOccurs="unbounded"/>
R      </xs:sequence>
R      <xs:attribute name="name" type="string50"/>
R      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
R      <xs:attribute name="count" type="xs:string"/>
R   </xs:complexType>
R
R
R
R   <!-- Абстрактный тип для типа struct or program. -->
R   <xs:element name="structOrProgram" abstract="true" />
R
R
R   <!-- Абстрактный тип для всех типов полей. -->
R   <xs:element name="stringFieldGroup" type="stringFieldType" abstract="true" />
```

```
R   <xs:element name="intFieldGroup" type="intFieldType" abstract="true" />
R   <xs:element name="unsignedIntFieldGroup" type="unsignedIntFieldType" abstract="true" />
R   <xs:element name="shortFieldGroup" type="shortFieldType" abstract="true" />
R   <xs:element name="unsignedShortFieldGroup" type="unsignedShortFieldType" abstract="true" />
R   <xs:element name="longFieldGroup" type="longFieldType" abstract="true" />
R   <xs:element name="zonedDecimalFieldGroup" type="zonedDecimalFieldType" abstract="true" />
R   <xs:element name="packedDecimalFieldGroup" type="packedDecimalFieldType" abstract="true" />
R   <xs:element name="floatFieldGroup" type="floatFieldType" abstract="true" />
R   <xs:element name="doubleFieldGroup" type="doubleFieldType" abstract="true" />
R   <xs:element name="hexBinaryFieldGroup" type="hexBinaryFieldType" abstract="true" />
R   <xs:element name="structFieldGroup" type="structFieldType" abstract="true" />
R
R
R
R   <!-- Объявление каждого элемента поля отдельным типом поля. -->
R   <xs:element name="stringField" type="stringFieldType" substitutionGroup="stringFieldGroup"
R               nillable="true"/>
R   <xs:element name="intField" type="intFieldType" nillable="true"
R               substitutionGroup="intFieldGroup" />
R   <xs:element name="unsignedIntField" type="unsignedIntFieldType"
R               substitutionGroup="unsignedIntFieldGroup" nillable="true"/>
R   <xs:element name="shortField" type="shortFieldType" nillable="true"
R               substitutionGroup="shortFieldGroup" />
R   <xs:element name="unsignedShortField" type="unsignedShortFieldType" nillable="true"
R               substitutionGroup="unsignedShortFieldGroup"  />
R   <xs:element name="longField" type="longFieldType" nillable="true"
R               substitutionGroup="longFieldGroup" />
R   <xs:element name="hexBinaryField" type="hexBinaryFieldType" nillable="true"
R               substitutionGroup="hexBinaryFieldGroup" />
R   <xs:element name="zonedDecimalField" type="zonedDecimalFieldType" nillable="true"
R               substitutionGroup="zonedDecimalFieldGroup"  />
R   <xs:element name="packedDecimalField" type="packedDecimalFieldType" nillable="true"
R               substitutionGroup="packedDecimalFieldGroup" />
R   <xs:element name="doubleField" type="doubleFieldType" nillable="true"
R               substitutionGroup="doubleFieldGroup" />
R   <xs:element name="floatField" type="floatFieldType" nillable="true"
R               substitutionGroup="floatFieldGroup" />
R
R
R   <xs:element name="structField" type="structFieldType" nillable="true"
R               substitutionGroup="structFieldGroup" />
R
R <!-- Параметр StructureField имеет один из следующих типов: stringField, intField,
R      shortField, longField, zonedDecimalField, packedDecimalField, floatField,
R      doubleField или hexBinaryField. -->
R       <xs:group name="structureField">
R         <xs:choice>
R           <xs:element ref="stringFieldGroup"/>
R           <xs:element ref="intFieldGroup"/>
R           <xs:element ref="unsignedIntFieldGroup"/>
R           <xs:element ref="shortFieldGroup"/>
R           <xs:element ref="unsignedShortFieldGroup"/>
R           <xs:element ref="longFieldGroup"/>
R           <xs:element ref="zonedDecimalFieldGroup"/>
R           <xs:element ref="packedDecimalFieldGroup"/>
R           <xs:element ref="floatFieldGroup"/>
R           <xs:element ref="doubleFieldGroup"/>
R           <xs:element ref="hexBinaryFieldGroup"/>
R           <xs:element ref="structParmGroup"/>
R           <xs:element ref="struct"/>
R         </xs:choice>
R       </xs:group>
R
R   <!-- Character field -->
R   <!-- Соответствие типам объектов AS400Text. -->
R    <xs:complexType name="stringFieldType">
R      <xs:simpleContent>
```

```
R         <xs:extension base="xs:string">
R           <!-- Число символов. -->
R           <xs:attribute name="length" type="xs:string"/>
R           <!-- Указывает CCSID поля на сервере. -->
R           <xs:attribute name="ccsid" type="xs:string"/>
R           <xs:attribute name="trim" type="trimType" />
R           <xs:attribute name="bytesPerChar" type="charType" />
R           <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
R           <xs:attributeGroup ref="commonFieldAttrs"/>
R         </xs:extension>
R       </xs:simpleContent>
R     </xs:complexType>
R
R
R   <!-- hexBinary field -->
R   <!-- Преобразуется в объект AS400ByteArray. -->
R     <xs:complexType name="hexBinaryFieldType">
R       <xs:simpleContent>
R         <xs:extension base="xs:hexBinary">
R           <xs:attributeGroup ref="commonFieldAttrs"/>
R         </xs:extension>
R       </xs:simpleContent>
R     </xs:complexType>
R
R
R   <!-- Float field -->
R   <!-- Преобразуется в объект AS400Float4. -->
R     <xs:complexType name="floatFieldType">
R       <xs:simpleContent>
R           <xs:extension base="xs:float">
R               <xs:attributeGroup ref="commonFieldAttrs"/>
R           </xs:extension>
R       </xs:simpleContent>
R      </xs:complexType>
R
R   <!-- zonedDecimal field -->
R   <!-- Преобразуется в объект AS400ZonedDecimal. -->
R     <xs:complexType name="zonedDecimalFieldType">
R       <xs:simpleContent>
R           <xs:extension base="zonedDecimal">
R               <xs:attributeGroup ref="commonFieldAttrs"/>
R           </xs:extension>
R       </xs:simpleContent>
R     </xs:complexType>
R
R   <!-- packedDecimal field -->
R   <!-- Преобразуется в объект AS400PackedDecimal. -->
R     <xs:complexType name="packedDecimalFieldType">
R       <xs:simpleContent>
R           <!-- В DDS значения binary имеют от 1 до 18 разрядов; если длина поля превосходит
R                greater than 9, then decimal positions value must be 0. -->
R           <xs:extension base="packedDecimal">
R               <xs:attributeGroup ref="commonFieldAttrs"/>
R           </xs:extension>
R       </xs:simpleContent>
R     </xs:complexType>
R
R   <!-- int field -->
R   <!-- Преобразуется в объект AS400Bin4. -->
R     <xs:complexType name="intFieldType">
R       <xs:simpleContent>
R           <xs:extension base="xs:int">
R              <xs:attributeGroup ref="commonFieldAttrs"/>
R           </xs:extension>
R       </xs:simpleContent>
R     </xs:complexType>
R
```

```
R   <!-- unsigned int field -->
R   <!-- Преобразуется в объект AS400Bin4. -->
R    <xs:complexType name="unsignedIntFieldType">
R      <xs:simpleContent>
R          <xs:extension base="xs:unsignedInt">
R             <xs:attributeGroup ref="commonFieldAttrs"/>
R          </xs:extension>
R      </xs:simpleContent>
R    </xs:complexType>
R
R   <!-- short field -->
R   <!-- Преобразуется в объект AS400Bin2. -->
R    <xs:complexType name="shortFieldType">
R      <xs:simpleContent>
R          <xs:extension base="xs:short">
R             <xs:attributeGroup ref="commonFieldAttrs"/>
R          </xs:extension>
R      </xs:simpleContent>
R    </xs:complexType>
R
R   <!-- unsigned short field -->
R   <!-- Преобразуется в объект AS400Bin2. -->
R    <xs:complexType name="unsignedShortFieldType">
R      <xs:simpleContent>
R          <xs:extension base="xs:unsignedShort">
R             <xs:attributeGroup ref="commonFieldAttrs"/>
R          </xs:extension>
R      </xs:simpleContent>
R    </xs:complexType>
R
R
R   <!-- long field -->
R   <!-- Преобразуется в объект AS400Bin8. -->
R    <xs:complexType name="longFieldType">
R      <xs:simpleContent>
R          <xs:extension base="xs:long">
R             <xs:attributeGroup ref="commonFieldAttrs"/>
R          </xs:extension>
R      </xs:simpleContent>
R    </xs:complexType>
R
R   <!-- double field -->
R   <!-- Преобразуется в объект AS400Float8. -->
R    <xs:complexType name="doubleFieldType">
R      <xs:simpleContent>
R          <xs:extension base="xs:double">
R             <xs:attributeGroup ref="commonFieldAttrs"/>
R          </xs:extension>
R      </xs:simpleContent>
R    </xs:complexType>
R
R
R
R   <!-- struct Field -->
R    <xs:complexType name="structFieldType">
R      <xs:simpleContent>
R        <xs:extension base="xs:string">
R          <xs:attribute name="struct" type="string50"/>
R          <xs:attributeGroup ref="commonFieldAttrs"/>
R        </xs:extension>
R      </xs:simpleContent>
R    </xs:complexType>
R
R
R <!-- Общие атрибуты для всех типов полей. -->
R <xs:attributeGroup name="commonFieldAttrs">
R    <xs:attribute name="name" type="string50"/>
```

```
R    <xs:attribute name="columnHeading1" type="string20"/>
R    <xs:attribute name="columnHeading2" type="string20"/>
R    <xs:attribute name="columnHeading3" type="string20"/>
R    <xs:attribute name="description" type="string50"/>
R    <xs:attribute name="defaultValue" type="xs:string"/><!-- Max length of string is 65535 characters. -->
R    <xs:attribute name="nullable" type="xs:boolean"/>
R    <xs:attribute name="isEmptyString" type="xs:boolean"/><!-- Indicate this is an empty string. -->
R </xs:attributeGroup>
R
R <!-- Служебные типы.  -->
R
R <xs:simpleType name="ccsidType">
R                     <xs:restriction base="xs:nonNegativeInteger">
R                      <xs:maxInclusive value="65535"/>
R                     </xs:restriction>
R                    </xs:simpleType>
R
R <xs:simpleType name="string10">
R                     <xs:restriction base="xs:string">
R                      <xs:maxLength value="10"/>
R                     </xs:restriction>
R                    </xs:simpleType>
R
R <xs:simpleType name="string20">
R                     <xs:restriction base="xs:string">
R                      <xs:maxLength value="20"/>
R                     </xs:restriction>
R                    </xs:simpleType>
R
R <xs:simpleType name="string50">
R                     <xs:restriction base="xs:string">
R                      <xs:maxLength value="50"/>
R                     </xs:restriction>
R                    </xs:simpleType>
R </xs:schema>
```

Синтаксис XPCML:

В схеме XPCML определяются несколько тегов элементов, и каждый из них содержит теги атрибутов.

В приведенной ниже таблице перечислены различные элементы, которые можно объявить и определить в исходном файле XPCML. Каждая запись в первом столбце связана с соответствующим разделом схемы XPCML.

Тег XPCML	**Описание**	**Эквивалентный тег PCML**
doubleParm	Определяет параметр типа double	данные (тип=float, длина=8)
arrayOfDoubleParm	Определяет параметр - массив элементов типа double	
floatParm	Определяет параметр типа float	данные (тип=float, длина=4)
arrayOfFloatParm	Определяет параметр - массив элементов типа float	
hexBinaryParm	Определяет параметр типа byte, представленный в шестнадцатеричном виде	byte (грубое соответствие, представлен в шестнадцатеричном виде)
arrayOfHexBinaryParm	Определяет массив элементов типа hexBinary	
intParm	Определяет параметр типа integer	данные (тип=int, длина=4)
arrayOfIntParm	Определяет параметр - массив элементов типа integer	

Тег XPCML	Описание	Эквивалентный тег PCML
longParm	Определяет параметр - массив элементов типа long	данные (тип=int, длина=8)
arrayOfLongParm	Определяет параметр - массив элементов типа long	
packedDecimalParm	Определяет параметр, содержащий упакованное десятичное значение	данные (тип=упакованное значение)
arrayOfPackedDecimalParm	Определяет параметр - массив упакованных десятичных значений	
parameterList	Указывает, что закрывающий тег представляет все определения параметров программы	
program	Открывает и закрывает тег XML, в котором описан вызов программы	программа
shortParm	Определяет параметр типа short	данные (тип int, длина 2)
arrayOfShortParm	Определяет параметр - массив элементов типа short	
stringParm	Определяет параметр типа string	
arrayOfStringParm	Определяет параметр - массив элементов типа string	
struct	Определяет именованную структуру, которую можно указать в качестве аргумента программы или в качестве поля другой именованной структуры	struct
arrayOfStruct	Определяет массив элементов типа struct	
structParm	Представляет ссылку на тег struct, который расположен в другой части документа XPCML и который необходимо включить в определенное расположение документа	данные (тип=struct)
arrayOfStructParm	Определяет массив элементов типа struct	
unsignedIntParm	Определяет параметр типа integer без знака	данные (тип=int, длина=4, точность=32)
arrayOfUnsignedIntParm	Определяет массив элементов типа integer без знака	
unsignedShortParm	Определяет параметр типа short без знака	данные (тип=int, длина=2, точность=16)
arrayOfUnsignedShortParm	Определяет массив элементов типа short без знака	
xpcml	Открывает и закрывает исходный файл XPCML, описывающий формат вызова программы	
zonedDecimalParm	Определяет параметр, имеющий зонное десятичное значение	данные (зонный тип)
arrayOfZonedDecimalParm	Определяет массив элементов зонного десятичного типа	

Атрибуты тегов XPCML:

В схеме XPCML определяются несколько тегов элементов, и каждый из них содержит теги атрибутов. В приведенной ниже таблице перечислены и описаны различные атрибуты каждого элемента:

Дополнительная подробная информация о тегах XPCML и их атрибутах приведена в разделе Схема XPCML.

Тег XPCML	Атрибут	Описание
hexBinaryParm	Позволяет заполнять последние 2 столбца данными и определять формат	
arrayOfHexBinaryParm		
doubleParm	Определяет параметр типа double	тип float (длины 8)
arrayOfDoubleParm	Определяет параметр - массив элементов типа double	
floatParm	Определяет параметр типа float	данные (тип float, длина 4)
arrayOfFloatParm	Определяет параметр- массив элементов типа float	
intParm	Определяет параметр типа integer	данные (тип int, длина 4)
arrayOfIntParm	Определяет параметр - массив элементов типа integer	
longParm	Определяет параметр - массив элементов типа long	данные (тип int, длина 8)
arrayOfLongParm	Определяет параметр- массив элементов типа long	
packedDecimalParm	Определяет параметр, содержащий упакованное десятичное значение	данные (упакованный тип)
arrayOfPackedDecimalParm	Определяет параметр - массив упакованных десятичных значений	
parameterList	Указывает, что закрывающий тег представляет все определения параметров программы	
программа	Открывает и закрывает тег XML, в котором описан вызов программы	
shortParm	Определяет параметр типа short	данные (тип int, длина 2)
arrayOfShortParm	Определяет параметр - массив элементов типа short	
stringParm	Определяет параметр типа string	
arrayOfStringParm	Определяет параметр - массив элементов типа string	
struct	Определяет именованную структуру, которую можно указать в качестве аргумента программы или в качестве поля другой именованной структуры	
arrayOfStruct	Определяет массив элементов типа struct	
structParm	Представляет ссылку на тег struct, который расположен в другой части документа XPCML и который необходимо включить в определенное расположение документа	данные (тип struct)

Тег XPCML	Атрибут	Описание
arrayOfStructParm	Определяет массив элементов типа struct	
unsignedIntParm	Определяет параметр типа integer без знака	данные (тип int, длина 4, точность 32)
arrayOfUnsignedIntParm	Определяет массив элементов типа integer без знака	
unsignedShortParm	Определяет параметр типа short без знака	данные (тип int, длина 2, точность 16)
arrayOfUnsignedShortParm	Определяет массив элементов типа short без знака	
xpcml	Открывает и закрывает исходный файл XPCML, описывающий формат вызова программы	
zonedDecimalParm	Определяет параметр, имеющий зонное десятичное значение	данные (зонный тип)
arrayOfZonedDecimalParm	Определяет массив элементов зонного десятичного типа	

Применение XPCML

Работа с языком XPCML имеет много общего с применением PCML. Ниже приведено общее описание основных этапов применения XPCML.

1. Описание спецификации вызова программы с помощью XPCML
2. Создание объекта ProgramCallDocument
3. Запуск программы с помощью метода ProgramCallDocument.callProgram()

Несмотря на многие общие черты с PCML, язык XPCML обладает расширенными функциональными возможностями:

- Автоматическая проверка значений параметров при анализе документа
- Указание и передача значений параметров программ
- Retrieve the results of a program call to your server in XPCML
- Преобразование документов PCML в эквивалентные документы XPCML
- Расширение схемы XPCML с помощью новых простых и сложных элементов и атрибутов

Например, IBM Toolbox for Java позволяет расширять схему XPCML с помощью новых параметров и типов данных. С помощью этой возможности XPCML можно уплотнять исходные файлы XPCML, что упрощает чтение файлов и работу с кодом.

Дополнительная информация о применении XPCML приведена на следующих страницах:

“Преобразование файлов PCML в XPCML” на стр. 443
Класс ProgramCallDocument содержит метод transformPCMLToXPCML, позволяющий преобразовывать существующие документы PCML в эквивалентные документы XPCML.

“Using XPCML to call a program on your server” на стр. 444
After you create your XPCML file, you need to create a ProgramCallDocument object that can use the XPCML specifications and data values to call a program on your System i5.

“Получение результатов вызова программы в формате XPCML” на стр. 445
После вызова программы на сервере вы можете с помощью метода ProgramCallDocument.getValue получить объекты Java, представляющие значения параметров программы.

“Передача значений параметров в формате XPCML” на стр. 445
Значения параметров программы можно задать в исходном файле XPCML и передать их в формате XPCML.

“Применение уплотненного XPCML” на стр. 446
Поскольку XPCML является расширяемым языком, вы можете определить новые типы параметров, расширяющие схему XPCML. При уплотнении XPCML схема XPCML расширяется за счет новых определений типов данных, которые упрощают и расширяют возможности чтения и применения документов XPCML.

“Идентификация ошибок анализатора в формате XPCML” на стр. 447
При проверке документов схемы XPCML анализатор XML, поддерживающий проверку полных схем, может создавать предупреждения и сообщения об исправимых и неисправимых ошибках при анализе.

Преобразование файлов PCML в XPCML:

Класс ProgramCallDocument содержит метод transformPCMLToXPCML, позволяющий преобразовывать существующие документы PCML в эквивалентные документы XPCML.

XPCML содержит определения, аналогичные определениям всех элементов и атрибутов языка PCML. Метод transformPCMLToXPCML() преобразует представление элементов и атрибутов PCML в эквивалентное представление XPCML.

Обратите внимание на то, что в некоторых случаях имена эквивалентных атрибутов XPCML отличаются от имен соответствующих имен в PCML. Например, атрибут usage в PCML соответствует атрибуту passDirection в XPCML. Дополнительная информация об отличиях в работе с XPCML и PCML приведена в разделе Схема и синтаксис XPCML .

Метод преобразует исходный документ PCML, переданный ему в виде объекта InputStream, в эквивалентный документ XPCML, который представлен объектом OutputStream. Поскольку метод transformPCMLToXPCML() является статическим, его можно вызвать без создания объекта ProgramCallDocument.

Пример: Преобразование документа PCML в документ XPCML

Ниже приведен пример преобразования документа PCML (с именем myPCML.pcml) в документ XPCML (с именем myXPCML.xpcml).

Примечание: В качестве расширения для файлов XPCML необходимо задать .xpcml. Это позволит классу ProgramCallDocument работать с этими файлами как с документами XPCML. Если это разрешение не будет задано, объект ProgramCallDocument будет интерпретировать их как исходные файлы PCML.

Документ PCML myPCML.pcml

```
<!-- myPCML.pcml -->
<pcml version="4.0">
   <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <data type="char" name="parm1" usage="in" passby="reference"
            minvrm="V5R2M0" ccsid="37" length="10" init="Value 1"/>
   </program>
</pcml>
```

Java code to convert myPCML.pcml to myPCML.xpcml

```
try {
  InputStream pcmlStream = new FileInputStream("myPCML.pcml");
  OutputStream xpcmlStream = new FileOutputStream("myXPCML.xpcml");
  ProgramCallDocument.transformPCMLToXPCML(pcmlStream, xpcmlStream);
}
```

```
catch (Exception e) {
  System.out.println("ошибка:  - "+e.getMessage());
  e.printStackTrace();
}
```

Полученный документ XPCML myXPCML.xpcml

```
<?xml version="1.0" encoding="UTF-8"?>
   <!-- myXPCML.xpcml -->
   <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">
   <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
      <parameterList>
         <stringParm name="parm1" passDirection="in" passMode="reference"
                     minvrm="V5R2M0" ccsid="37" length="10">Value 1
         </stringParm>
      </parameterList>
   </program>
</xpcml>
```

Дополнительная информация о методе transformPCMLToXPCML() и классе ProgramCallDocument приведена на следующей странице:

Справочная информация Java для класса ProgramCallDocument

Using XPCML to call a program on your server:

After you create your XPCML file, you need to create a ProgramCallDocument object that can use the XPCML specifications and data values to call a program on your System i5.

Для создания объекта XPCML ProgramCallDocument необходимо указать имя файла XPCML в качестве входных данных конструктора ProgramCallDocument. При создании объекта XPCML ProgramCallDocument с помощью этого способа сначала анализируется и проверяется документ XPCML, после чего создается объект ProgramCallDocument.

Перед анализом и проверкой документа XPCML убедитесь, что переменная CLASSPATH содержит анализатор XML, поддерживающий проверку полных схем. Дополнительная информация о требованиях для запуска XPCML приведена на следующей странице:

“Требования для применения XPCML” на стр. 421

Ниже приведен пример создания объекта ProgramCallDocument для файла XPCML myXPCML.xpcml.

```
system = new AS400();
// Создание объекта ProgramCallDocument, в котором будет выполняться анализ файла.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "myXPCML.xpcml");
```

Единственное различие между созданием объекта XPCML ProgramCallDocument и объекта PCML ProgramCallDocument заключается в типе объекта (XPCML или PCML), передаваемого конструктору.

Примечание: В качестве расширения для файлов XPCML необходимо задать .xpcml. Это позволит классу ProgramCallDocument работать с этими файлами как с документами XPCML. Если вы не укажете это расширение, то объект ProgramCallDocument будет интерпретировать их как исходные файлы PCML.

Using XPCML to call a program on your server

R После того, как объект ProgramCallDocument создан, для работы с документом XPCML можно применять
R любые методы класса ProgramCallDocument. For example, call a System i program by using

R ProgramCallDocument.callProgram() or change the value for an XPCML input parameter before calling the server
R program by using the appropriate ProgramCallDocument.setValue method.

Ниже приведен пример создания объекта ProgramCallDocument для файла XPCML (с именем myXPCML.xpcml). После создания объекта ProgramCallDocument в примере вызывается программа (PROG1), заданная в документе XPCML. В этой ситуации применение файлов XPCML отличается от применения файлов PCML только тем, что в примере конструктору ProgramCallDocument передается объект XPCML.

После чтения и анализа документа XPCML он выполняет те же функции, что и документ PCML. Документ XPCML может при этом использовать любые методы PCML.

```
system = new AS400();

// Создание объекта ProgramCallDocument, в котором будет выполняться анализ файла.
ProgramCallDocument xpcmlDoc = new ProgramCallDocument(system, "myXPCML.xpcml");

// Вызов программы PROG1
boolean rc = xpcmlDoc.callProgram("PROG1");
```

Получение результатов вызова программы в формате XPCML:

После вызова программы на сервере вы можете с помощью метода ProgramCallDocument.getValue получить объекты Java, представляющие значения параметров программы.

Кроме того, приведенные ниже методы generateXPCML класса ProgramCallDocument позволяют получить результаты работы программы в формате XPCML:

- generateXPCML(String fileName): создает результаты в формате XPCML для всего исходного файла XPCML, с помощью которого был создан объект ProgramCallDocument. Сохраняет документ XPCML в файле с указанным именем.
- generateXPCML(String pgmName, String fileName): создает результаты в формате XPCML только для указанной программы и ее параметров. Сохраняет документ XPCML в файле с указанным именем.
- generateXPCML(java.io.OutputStream outputStream): создает результаты в формате XPCML для всего исходного файла XPCML. Сохраняет документ XPCML в указанном объекте OutputStream.
- generateXPCML(String pgmName, java.io.OutputStream outputStream): создает результаты в формате XPCML только для указанной программы и ее параметров. Сохраняет документ XPCML в указанном объекте OutputStream.

For more information about the ProgramCallDocument class, see the ProgramCallDocument Javadoc information.

R The following example shows how you can construct an XPCML ProgramCallDocument, call a System i program, and
R retrieve the results of the program call as XPCML.

“Пример: Получение результатов вызова программы в виде файла XPCML” на стр. 755

Информация, связанная с данной

ProgramCallDocument Javadoc

Передача значений параметров в формате XPCML:

Значения параметров программы можно задать в исходном файле XPCML и передать их в формате XPCML.

При чтении и анализе документа XPCML класс ProgramCallDocument автоматически вызывает метод setValue для каждого параметра, указанного в файле XPCML.

Передача значений параметров в формате XPCML позволяет обойтись без создания кода Java, задающего значения сложных структур и массивов.

Ниже приведены несколько примеров, демонстрирующие различные способы создания массивов и передачи значений параметров в формате XPCML:

“Пример: Передача значений параметров в виде файла XPCML” на стр. 758

“Пример: Передача массивов значений параметров в виде файла XPCML” на стр. 759

Применение уплотненного XPCML:

Поскольку XPCML является расширяемым языком, вы можете определить новые типы параметров, расширяющие схему XPCML. При уплотнении XPCML схема XPCML расширяется за счет новых определений типов данных, которые упрощают и расширяют возможности чтения и применения документов XPCML.

Приведенные ниже сведения предназначены для пользователей, знакомых со схемой XPCML. Дополнительная информация о схеме XPCML приведена на следующей странице:

“Схема и синтаксис XPCML” на стр. 422

Для уплотнения исходного файла XPCML применяется метод ProgramCallDocument.condenseXPCML, который создает следующие объекты:

- Расширенную схему, которая содержит новые определения типов для каждого параметра в существующем исходном файле XPCML
- Новый исходный файл New XPCML, использующий определения типов, описанные в расширенной схеме

Дополнительная информация об уплотнении исходных файлов XPCML приведена на следующих страницах:

“Уплотнение существующих документов XPCML”

“Пример: Создание объекта ProgramCallDocument с помощью уплотненного файла XPCML” на стр. 764

“Пример: Получение результатов вызова программы в виде уплотненного файла XPCML” на стр. 764

Уплотнение существующих документов XPCML:

Сжатие существующих документов XPCML позволяет получить исходные файлы XPCML, чтение и применение которых значительно проще стандартных. Для создания уплотненных файлов XPCML применяется метод ProgramCallDocument.condenseXPCML.

Для его вызова необходимо указать следующие параметры:

- Поток входных данных, представляющих существующий файл XPCML
- Поток выходных данных, представляющий уплотненный XPCML
- Поток выходных данных, представляющих новую расширенную схему
- Имя новой схемы в соответствующем формате (например, mySchema.xsd)

For more information about condenseXPCML() and the ProgramCallDocument class, see the ProgramCallDocument Javadoc information.

ProgramCallDocument.condenseXPCML() - это статический метод, т.е. для его вызова не нужно создавать объект ProgramCallDocument.

Примеры

Ниже приведены примеры уплотнения документов XPCML.

Первый пример достаточно прост и содержит исходный документ XPCML, полученный уплотненный документ XPCML и расширенную схему. Второй пример длиннее и сложнее - он содержит также код Java, в котором вызывается метод condenseXPCML(), и некоторые новые определения типов в расширенной схеме:

“Пример: Уплотнение существующего документа XPCML” на стр. 761

“Пример: Уплотнение существующего документа XPCML с исходным кодом Java” на стр. 761

Информация, связанная с данной

ProgramCallDocument Javadoc

Идентификация ошибок анализатора в формате XPCML:

При проверке документов схемы XPCML анализатор XML, поддерживающий проверку полных схем, может создавать предупреждения и сообщения об исправимых и неисправимых ошибках при анализе.

Предупреждения и исправимые ошибки анализатора позволяют продолжить анализ документа. Просмотрев их, вы можете определить причины неполадок при анализе исходного файла XPCML. Неисправимые ошибки приводят к прекращению обработки файла и созданию исключительной ситуации.

Для просмотра предупреждений и исправимых ошибок анализатора необходимо включить трассировку в соответствующем приложении и установить категорию трассировки PCML.

Пример

Анализатор XML, поддерживающий проверку полных схем, создает сообщение об ошибке с информацией о том, что числовому параметру не присвоено значение. Ниже приведен пример исходного кода XPCML и возникшей в результате его обработки исправимой ошибки анализатора:

Исходный файл XPCML

```
<program name="prog1"/>
   <parameterList>
      <intParm name="parm1"/>
   </parameterList>
</program>
```

Полученная ошибка

```
Четверг 25 марта 15:21:44 CST 2003  [Ошибка]: cvc-complex-type.2.2: У элемента
intParm не должно быть элементов [дочерних объектов]
и для него должно быть указано допустимое значение.
```

Для того чтобы такие ошибки не заносились в протокол, необходимо добавить атрибут nil=true в элемент intParm. Этот атрибут указывает анализатору на то, что этому элементу намеренно не присвоено значение. Ниже приведена предыдущая версия исходного файла XPCML с атрибутом nil=true:

```
<program name="prog1"/>
   <parameterList>
      <intParm xsi:nil="true" name="parm1"/>
   </parameterList>
</program>
```

Часто задаваемые вопросы (FAQ)

IBM Toolbox for Java frequently asked questions (FAQs) provide answers to questions about optimizing your IBM Toolbox for Java performance, performing troubleshooting, using JDBC, and more.

- IBM Toolbox for Java FAQ : Find answers to many types of questions, including improving performance, using i5/OS, performing troubleshooting, and more.

- IBM Toolbox for Java JDBC FAQ : Find answers to questions about using JDBC with IBM Toolbox for Java.

Советы программисту

В этом разделе приведен ряд советов по работе с IBM Toolbox for Java.

Завершение работы программы на Java

Для корректного завершения работы программы на Java последней должна вызываться функция System.exit(0).

Примечание: Не применяйте функцию System.exit(0) в сервлетах, поскольку при этом будет завершена работа виртуальной машины Java.

Для подключения к серверу в IBM Toolbox for Java применяются пользовательские нити. В связи с этим, если метод System.exit(0) не будет вызван, программа на Java может быть завершена некорректно.

Вызов функции System.exit(0) является не обязательным требованием, а рекомендацией. В некоторых случаях эта функция программы на Java необходима, а во всех остальных случаях ее вызов не приведет к ошибке.

Пути к объектам интегрированной файловой системы сервера

Для работы с объектами сервера - программами, библиотеками, командами и буферными файлами - в программе на Java должны применяться имена объектов интегрированной файловой системы (пути). The integrated file system name is the name of a server object as it might be accessed in the library file system of the System i5 integrated file system.

Путь может содержать следующие компоненты:

Компонент пути	**Описание**
библиотека	Библиотека, в которой находится объект. Библиотека является **обязательным** элементом пути. Имя библиотеки состоит из не более чем 10 символов, за которыми следует расширение **.lib**.
объект	Имя объекта, на который указывает путь в интегрированной файловой системе. Объект является **обязательным** элементом пути. Имя объекта состоит из не более чем 10 символов, за которыми следует расширение вида **.тип**, обозначающее тип объекта. Тип объекта указывается в параметре OBJTYPE команд управляющего языка (CL), например, команды Работа с объектам (WRKOBJ).
тип	Тип объекта. Тип обязателен, если задан **объект**. (См. выше описание компонента пути **объект**.) Тип состоит из не более чем 6 символов.
элемент	Имя элемента, на который указывает путь в интегрированной файловой системе. Элемент является **необязательным** компонентом пути. Его можно задать, только если **тип объекта - FILE**. Имя элемента состоит из не более чем 10 символов, за которые следует расширение **.mbr**.

При определении и использовании имени объекта интегрированной файловой системы необходимо следовать приведенным ниже правилам:

- Разделителем компонентов пути служит косая черта (/).
- Структура библиотек файловой системы сервера хранится в корневом каталоге QSYS.LIB.
- Имена объектов, расположенных в библиотеке QSYS сервера, имеют следующий формат:

  ```
  /QSYS.LIB/объект.тип
  ```
- Имена объектов, расположенных в других библиотеках, имеют следующий формат:

  ```
  /QSYS.LIB/библиотека.LIB/объект.тип
  ```
- Расширение объекта - это принятое на сервере сокращение для обозначения типа объекта.

Для просмотра списка допустимых типов введите команду CL, у которой есть параметр Тип объекта, и нажмите **F4** (Приглашение) в поле Тип. Например, вызовите команду Работа с объектами (WRKOBJ).

В приведенной ниже таблице указаны некоторые часто используемые типы объектов и соответствующие сокращения:

Тип объекта	**Сокращение**
команда	.CMD
очередь данных	.DTAQ
файл	.FILE
ресурс шрифта	.FNTRSC
определение формы	.FORMDF
библиотека	.LIB
элемент	.MBR
перекрытие	.OVL
определение страницы	.PAGDFN
сегмент страницы	.PAGSET
программа	.PGM
очередь вывода	.OUTQ
буферный файл	.SPLF

При определении имени объекта в интегрированной файловой системе можно руководствоваться следующими описаниями:

Имя в интегрированной файловой системе	**Описание**
/QSYS.LIB/MY_LIB.LIB/MY_PROG.PGM	Программа MY_PROG в библиотеке MY_LIB на сервере
/QSYS.LIB/MY_LIB.LIB/MY_QUEUE.DTAQ	Очередь данных MY_QUEUE в библиотеке MY_LIB на сервере
/QSYS.LIB/YEAR1998.LIB/MONTH.FILE/JULY.MBR	Элемент JULY файла MONTH в библиотеке YEAR1998 на сервере

Специальные значения интегрированной файловой системы

R Various IBM Toolbox for Java classes recognize special values in integrated file system path names. The traditional
R format for these special values (as used on an i5/OS command line) begins with an asterisk (***ALL**). However, in a
R Java program that uses IBM Toolbox for Java classes, the format for these special values begins and ends with percent
R signs (**%ALL%**).

Примечание: В интегрированной файловой системе символом подстановки является звездочка.

The following table shows which of these special values the IBM Toolbox for Java classes recognize for particular path name components. The table also shows how the traditional format for these special values differ from the format used in IBM Toolbox for Java classes.

Компонент пути	Обычный формат	Формат IBM Toolbox for Java
Имя библиотеки	*ALL	%ALL%
	*ALLUSR	%ALLUSR%
	*CURLIB	%CURLIB%
	*LIBL	%LIBL%
	*USRLIBL	%USRLIBL%
Имя объекта	*ALL	%ALL%
Имя элемента	*ALL	%ALL%
	*FILE	%FILE%
	*FIRST	%FIRST%
	*LAST	%LAST%

Информация о построении и синтаксическом анализе имен объектов интегрированной файловой системы приведена в описании класса QSYSObjectPathName.

Дополнительная информация об интегрированной файловой системе приведена в разделе Интегрированная файловая система - Основная информация.

Managing connections in Java programs

У пользователя должна быть возможность создавать, запускать и завершать соединения с сервером. Ниже обсуждаются основные принципы управления соединениями с сервером, а также приводятся некоторые примеры программного кода.

R To connect to a System i5, your Java program must create an AS400 object. The AS400 object contains up to one
R socket connection for each System i5 server type. Каждая служба является заданием сервера iSeries и
R предоставляет доступ к данным этого сервера.

Примечание: При создании объектов Enterprise JavaBean (EJB) необходимо следовать спецификации EJB, не позволяющей создавать нити во время подключения. Это обязательное требование, несмотря на то, что отказ от поддержки нитей IBM Toolbox for Java может снизить производительность приложения.

R Every connection to each server has its own job on the system. Большинство серверов поддерживают следующие
R службы:

- JDBC
- Вызов программ и команд
- Интегрированная файловая система
- Печать
- Очередь данных
- Доступ на уровне записей

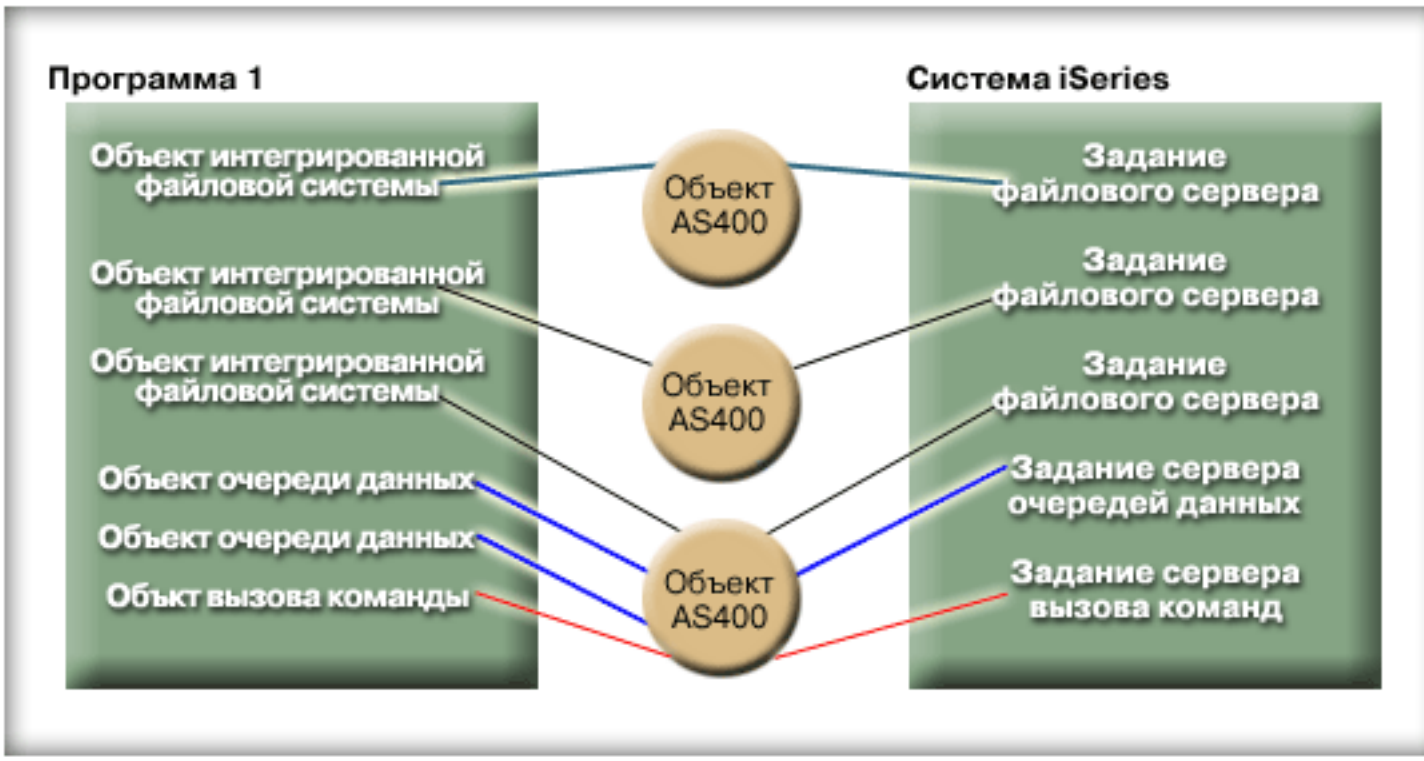
Примечание:

- Класс print создает для каждого объекта AS400 одно соединение с сокетом в случае, если приложение не отправляет сразу два запроса к серверу сетевой печати.

- При необходимости класс print создает дополнительное соединение с сокетом сервера сетевой печати. Если дополнительное соединение простаивает в течение 5 минут, оно прерывается.

R The Java program can control the number of connections to the system. To optimize communications performance, a
R Java program can create multiple AS400 objects for the same server as shown in Figure 1. This creates multiple socket
R connections to the system.

Figure 1: Java program creating multiple AS400 objects and socket connections for the same system



R Для минимизации объема ресурсов сервера iSeries, затрачиваемого на обмен данными, создайте только один
R объект AS400, как показано на рис. 2. Это позволяет сократить число соединений и объем занятых ресурсов
R системы.

Figure 2: Java program creating a single AS400 object and socket connection for the same system

Примечание: Несмотря на то, что при использовании большого числа соединений повышается объем занятых ресурсов сервера, дополнительные соединения могут обеспечивать определенные преимущества. Наличие нескольких соединений позволяет программе на Java выполнять обработку параллельно, что ускоряет работу приложения.

You can also choose to use a connection pool to manage connections as shown in Figure 3. This approach reduces the amount of time it takes to connect by reusing a connection previously established for the user.

Figure 3: Java program getting a connection from an AS400ConnectionPool to a server

Ниже приведены примеры создания и применения объектов AS400:

Пример 1: В приведенном ниже примере создаются два объекта CommandCall, которые отправляют команды на один сервер. Поскольку объекты CommandCall работают с одним и тем же объектом AS400, будет создано только одно соединение с сервером.

```
            // Создание объекта AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

            // Создание двух объектов вызова команды,
            // работающих с одним и тем же объектом AS400.
     CommandCall cmd1 = new CommandCall(sys,"myCommand1");
     CommandCall cmd2 = new CommandCall(sys,"myCommand2");

            // Запуск команд.  При первом запуске команды создается
            // соединение. Поскольку обе команды обрабатываются одним
// объектом AS400, второй объект команды будет применять
            // соединение, установленное первой командой.
     cmd1.run();
     cmd2.run();
```

Example 2: In the following example, two CommandCall objects are created that send commands to the same system. Объекты CommandCall работают с разными объектами AS400, поэтому будет создано два соединения с системой.

```
            // Создание двух объектов AS400 для одного сервера.
     AS400 sys1 = new AS400("mySystem.myCompany.com");
     AS400 sys2 = new AS400("mySystem.myCompany.com");

            // Создание двух объектов вызова команды.
            // Они применяют разные объекты AS400.
     CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
     CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

            // Запуск команд.  При первом запуске команды создается
            // соединение. Поскольку второй объект команды
// применяет другой объект AS400, при запуске второй
            // команды также устанавливается соединение.
     cmd1.run();
     cmd2.run();
```

Пример 3: В этом примере создаются объекты CommandCall и IFSFileInputStream, работающие с одним объектом AS400. Because the CommandCall object and the IFSFileInput Stream object use different services on the server, two connections are created.

```
        // Создание объекта AS400.
AS400 newConn1 = new AS400("mySystem.myCompany.com");

        // Создание объекта вызова команды.
CommandCall cmd = new CommandCall(newConn1,"myCommand1");

        // Создание файлового объекта. При этом
        // AS400 подключится к службе файлов.
IFSFileInputStream file = new IFSFileInputStream(newConn1,"/myfile");

        // Запуск команды. Создается
        // соединение со службой команд.
cmd.run();
```

Example 4: In the following example, an AS400ConnectionPool is used to get a System i5 connection. В этом примере (как и в Примере 3) не указана служба, поэтому при запуске команды будет установлено соединение со службой выполнения команд.

```
        // Создание пула AS400ConnectionPool.
AS400ConnectionPool testPool1 = new AS400ConnectionPool();
        // Создание соединения.
AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID", "myPassword");
        // Создание объекта вызова команды для объекта AS400.
CommandCall cmd = new CommandCall(newConn1,"myCommand1");
        // Запуск команды. Создается
        // соединение со службой команд.
cmd.run();
        // Возврат соединения в пул.
testPool1.returnConnectionToPool(newConn1);
```

Пример 5:В этом примере с помощью при запросе соединения из пула AS400ConnectionPool устанавливается соединение с указанной службой. В результате при выполнении команды не будет тратиться время на установление соединения (см. Пример 4). Если соединение было возвращено в пул, то в ответ на следующий запрос из пула может быть получен тот же объект соединения. Это означает, что ни при создании, ни при использовании соединения не расходуется дополнительное время.

```
        // Создание пула AS400ConnectionPool.
AS400ConnectionPool testPool1 = new AS400ConnectionPool();
        // Создание соединения с службой AS400.COMMAND. (Необходимо использовать ограничения
        // на номер службы, определенные в классе AS400 (FILE, PRINT, COMMAND, DATAQUEUE и т.д.))
AS400 newConn1 = testPool1.getConnection("myAS400", "myUserID", "myPassword", AS400.COMMAND);
        // Создание объекта вызова команды для объекта AS400.
CommandCall cmd = new CommandCall(newConn1,"myCommand1");
        // Запуск команды. Соединение со службой выполнения
        // команд уже установлено.
cmd.run();
        // Возврат соединения в пул.
testPool1.returnConnectionToPool(newConn1);
        // Настройка другого соединения со службой выполнения команд.
        // В этом случае будут возвращены
соединения, что и в предыдущем
        // примере (не требуется дополнительное время работы соединений
        // при использовании службы команд или отправке запроса.
AS400 newConn2 = testPool1.getConnection("myAS400", "myUserID", "myPassword", AS400.COMMAND);
```

Подключение и отключение

Программа на Java может устанавливать и прерывать соединения. По умолчанию соединение устанавливается тогда, когда нужно получить данные с сервера iSeries. Время создания соединения можно задать, установив предварительное соединение с сервером с помощью метода connectService() объекта AS400.

Объект AS400ConnectionPool позволяет создать соединение с определенной службой без вызова метода connectService(), как показано в вышеприведенном Примере 5.

The following examples show Java programs connecting to and disconnecting from the system.

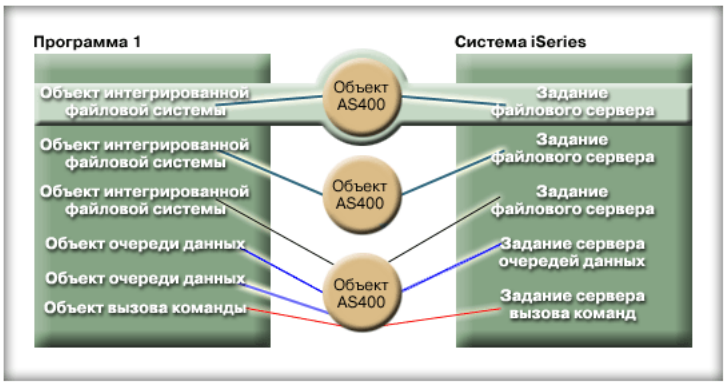
Example 1: This example shows how to preconnect to the system:

```
            // Создание объекта AS400.
    AS400 system1 = new AS400("mySystem.myCompany.com");

            // Подключение к службе команд. Это
// выполняется до того, как впервые
            // потребуется отправить данные этой службе.
// Если вы этого не сделаете явно, то объект AS400
                      // автоматически установит соединение.
    system1.connectService(AS400.COMMAND);
```

Пример 2: Программа на Java должна вовремя прервать созданное соединение. Это выполняется либо неявно - объектом AS400, либо явно - путем указания оператора в программе на Java. Для отключения программы на Java нужно вызвать метод disconnectService() для объекта AS400. Для повышения производительности рекомендуется отключать программу на Java только тогда, когда она закончит работать со службой. Если программа на Java будет отключена раньше, объект AS400 вновь попытается установить соединение, когда потребуется получить данные от службы.

На рис. 4 показано, что разрыв первого соединения с объектом интегрированной файловой системы не влечет за собой разрыв всех остальных соединений с объектами интегрированной файловой системы.

Рис. 4: Отключение одного объекта, работающего со службой своего экземпляра объекта AS400



Ниже приведен пример прерывания соединения программы на Java:

```
            // Создание объекта AS400.
    AS400 system1 = new AS400("mySystem.myCompany.com");

            // ... вызов нескольких команд сервера.
            // Поскольку метод connectService() не
// был вызван, объект AS400 автоматически
            // подключается при запуске первой команды.

            // Все команды выполнены, поэтому соединение
            // завершается.
    system1.disconnectService(AS400.COMMAND);
```

Пример 3: Соединение применяется несколькими объектами, которые обращаются к одной и той же службе и работают с одним объектом AS400. Прерывание соединения приводит к отключению всех объектов, работающих с одной и той же службой через один экземпляр объекта AS400, как показано на рис. 5.

Рис. 5: Отключение всех объектов, работающих с одной и той же службой через общий объект AS400

Например, два объекта CommandCall работают с одним объектом AS400. Метод disconnectService() прервет соединение с обоими объектами CommandCall. При вызове метода run() для второго объекта CommandCall объекту AS400 придется еще раз установить соединение:

```
            // Создание объекта AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

            // Создание двух объектов вызова команды.
     CommandCall cmd1 = new CommandCall(sys,"myCommand1");
     CommandCall cmd2 = new CommandCall(sys,"myCommand2");

            // Запуск первой команды
     cmd1.run();

            // Отключение от службы команд.      sys.disconnectService(AS400.COMMAND);

            // Запуск второй команды. Объект AS400 должен
// повторно подключиться к серверу.
     cmd2.run();

            // Отключение от службы команд.              // она не понадобится.
     sys.disconnectService(AS400.COMMAND);
```

Пример 4: Не все классы IBM Toolbox for Java поддерживают автоматическое восстановление соединения. Некоторые методы классов интегрированной файловой системы не восстанавливают соединение. Это связано с тем, что за время, пока соединение прервано, другой процесс может удалить или изменить файл. В следующем примере два объекта файла работают с одним объектом AS400. При вызове метода disconnectService() прерывается соединение с обоими объектами. Метод read() объекта IFSFileInputStream не будет выполнен, так как соединение с сервером прервано.

```
            // Создание объекта AS400.
     AS400 sys = new AS400("mySystem.myCompany.com");

            // Создание двух объектов файла. При создании
// первого объекта устанавливается соединение с сервером.
            // Второй объект применяет соединение, созданное
// первым объектом.
     IFSFileInputStream file1 = new IFSFileInputStream(sys,"/file1");
     IFSFileInputStream file2 = new IFSFileInputStream(sys,"/file2");

            // Чтение данных из первого файла и закрытие этого файла.
     int i1 = file1.read();
     file1.close();

            // Отключение от службы файлов.      sys.disconnectService(AS400.FILE);

            // Данные из второго файла не будут прочитаны,             // так как соединение со службой файлов
            // прервано. Программа должна позже восстановить
```

```
// соединение, либо создать другой объект
            // AS400 для второго файла (в этом случае с ним
            // будет установлено отдельное соединение).
    int i2 = file2.read();

            // Закрытие второго файла.
    file2.close();

            // Отключение от службы файлов.              // она не понадобится.
    sys.disconnectService(AS400.FILE);
```

i5/OS Java virtual machine

Классы IBM Toolbox for Java выполняются на виртуальной машине Java (JVM) продукта IBM Developer Kit for Java (i5/OS).

В действительности классы могут применяться на любой платформе, поддерживающей спецификации Java 2 Software Development Kit (J2SDK).

R For more information about System i support for different Java platforms, see Support for multiple JDKs.

Comparing the i5/OS Java virtual machine and the IBM Toolbox for Java classes

You always have at least two ways to access a server resource when your Java program is running on the IBM Developer Kit for Java (i5/OS) Java virtual machine (JVM).

с помощью одного из следующих интерфейсов:

- Встроенные функции Java
- С помощью класса IBM Toolbox for Java

При выборе интерфейса учтите следующие особенности:

- **Расположение** - Выбор интерфейса в значительной степени зависит от того, в какой системе будет выполняться программа. Возможны следующие варианты:

R – Программа запускается только на клиенте

R – Программа запускается только на сервере

R – Run on both client and server, but in both cases the resource is a System i resource?

R – Run on one i5/OS JVM and access resources on another System i?

R – Программа запускается на серверах различных типов

R If the program runs on both client and server (including a System i as a client to a second System i) and accesses
R only System i resources, it may be best to use the IBM Toolbox for Java interfaces.

Если программа обращается к серверам различных типов, то рекомендуется использовать стандартные интерфейсы Java.

R • **Consistency / Portability** - The ability to run IBM Toolbox for Java classes on System i means that the same
R interfaces can be used for both client programs and server programs. В этом случае вам не потребуется изучать
R два набора интерфейсов, что ускорит вашу работу.

R Однако программы, в которых применяются интерфейсы IBM Toolbox for Java, будут поддерживаться
R только одним типом **серверов**.

R If your program must run to anSystem i as well as other servers, you may find it better to use the facilities that are
R built into Java.

R • **Complexity** - The IBM Toolbox for Java interface is built especially for easy access to a System i resource. Чаще
R всего вместо классов IBM Toolbox for Java вам придется создавать собственные программы,
R обращающиеся к ресурсам и взаимодействующие с основной программой через Стандартный интерфейс
R Java (JNI).

R Решите, что лучше: создавать собственные программы, применяющие стандартный интерфейс Java, или
R воспользоваться интерфейсом IBM Toolbox for Java в ущерб переносимости.

- **Доступные функции** - В целом интерфейс IBM Toolbox for Java предоставляет более широкий набор функций по сравнению со стандартным интерфейсом Java. Например, класс IFSFileOutputStream лицензионной программы IBM Toolbox for Java содержит больше функций, чем класс FileOutputStream из пакета java.io. Using IFSFileOutputStream makes your program specific to System i, however. При применении класса IBM Toolbox for Java вы не сможете писать **переносимые** программы.

  Решите, что важнее: переносимость или возможность пользоваться дополнительными средствами.
- **Ресурсы** - Если программа выполняется Виртуальной машиной Java для i5/OS, то многие классы IBM Toolbox for Java отправляют запросы через серверы хоста. Следовательно, запросы на доступ к ресурсам обрабатываются вторым заданием (заданием сервера).

  Для обработки таких запросов требуется больше ресурсов, чем для выполнения стандартных методов Java, работающих под управлением задания программы на Java.
- **System i as a client** - If your program runs on one System i and accesses data on a second System i , your best choice may be to use IBM Toolbox for Java classes. These classes provide easy access to the resource on the second System i.

  Примером может служить очередь данных. Интерфейсы очереди данных лицензионной программы IBM Toolbox for Java обеспечивают доступ к ресурсу очереди данных.

  Using the IBM Toolbox for Java also means your program works on both a client and server to access a data queue on an System i. It also works when running on one System i to access a data queue on another System i.

  Альтернативой служит создание отдельной программы (например, на языке C), которая обращается к очереди данных. Программа на Java будет вызывать эту программу для обращения к очереди данных.

  Программа на Java, созданная таким способом, будет переносимой, если вы создадите одну версию основной программы и несколько версий методов для обращения к очередям данных серверов различных типов.

Running IBM Toolbox for Java classes on the i5/OS Java virtual machine

There are some special considerations for running the IBM Toolbox for Java classes on the IBM Developer Kit for Java (i5/OS) Java virtual machine (JVM).

Вызов команд

Существует два стандартных способа вызова команд:

- The IBM Toolbox for Java CommandCall class
- С помощью метода java.lang.Runtime.exec

Класс CommandCall создает список сообщений, который может просмотреть программа на Java после выполнения команды. Метод java.lang.runtime.exec() не создает такой список сообщений.

Метод java.lang.Runtime.exec можно использовать на различных платформах, поэтому, если приложение будет обращаться к файлам на серверах различных типов, рекомендуется применять этот метод.

Интегрированная файловая система

Common ways to access a file in the System i5 integrated file system:

- The IFSFile classes of the IBM Toolbox for Java licensed program
- С помощью классов файлов из пакета java.io

The IBM Toolbox for Java integrated file system classes have the advantage of providing more function than the java.io classes. The IBM Toolbox for Java classes also work in applets, and they do not need a method of redirection (such as System i Access for Windows) to get from a workstation to the server.

Преимуществом классов java.io является то, что они поддерживаются многими платформами. Их рекомендуется применять в тех программах, которые будут обращаться к серверам различных типов.

L If you use java.io classes on a client, you need a method of redirection (such as the System i Access for Windows) to get to the server file system.

JDBC

Two IBM-supplied JDBC drivers are available to programs running on the i5/OS JVM:

- The IBM Toolbox for Java JDBC driver
- The IBM Developer Kit for Java JDBC driver

The IBM Toolbox for Java JDBC driver is best to use when the program is running in a client/server environment.

R The IBM Developer Kit for Java JDBC driver is best to use when the program is running on the server.

Если программа выполняется и на рабочей станции, и на сервере, то имя драйвера должно не задаваться в программе, а считываться из системного значения.

Вызов программ

Существует два стандартных способа вызова программ:

- The ProgramCall class of the IBM Toolbox for Java
- Посредством Стандартного интерфейса Java (JNI)

R The ProgramCall class of the IBM Toolbox for Java licensed program has the advantage that it can call any server
R program.

R You may not be able to call your server program through JNI. Однако JNI поддерживается большим числом
R платформ.

Setting system name, user ID, and password with an AS400 object in the i5/OS Java virtual machine

The AS400 object allows special values for system name, user ID, and password when the Java program is running on the IBM Toolbox for Java (i5/OS) Java virtual machine (JVM).

When you run a program on the i5/OS JVM, be aware of some special values and other considerations:

- Если программа выполняется на сервере, то приглашение на ввод ИД пользователя и пароля не выдается. Дополнительная информация о выборе ИД пользователя и пароля в среде сервера приведена в разделе ИД пользователя и пароль в объекте AS400 - Обзор.
- Если имя системы, ИД пользователя и пароль не заданы в объекте AS400, компьютер будет подключен к текущему серверу. При этом будет указан ИД пользователя и пароль задания, запустившего программу на Java. При подключении к виртуальной машине версии v4r4 или выше этот пароль пользователя можно расширить, так же, как и все остальные компоненты IBM Toolbox for Java.
- В качестве имени системы можно указать специальное значение **localhost**. В этом случае объект AS400 подключится к текущему серверу.
- В качестве ИД пользователя или пароля в объекте AS400 можно указать специальное значение ***current**. Оно означает, что должен применяться ИД пользователя или пароль задания, запустившего программу на Java. Дополнительная информация о специальном значении *current приведена в примечаниях.
- The special value, ***current**, can be used as the user ID or password on the AS400 object when the Java program is running on the i5/OS JVM of one server, and the program is accessing resources on another System i. В этом случае при подключении к целевой системе будет применяться ИД пользователя и пароль задания, запустившего программу на Java в исходной системе. Дополнительная информация о специальном значении *current приведена в примечаниях.

  Примечания:

- Пароль ″*current″ недопустим, если программа на Java обращается к отдельным записям файла в версии V4R3 или ниже. В этом случае можно задать имя системы ″localhost″ и ИД пользователя ″*current″; программа на Java должна будет предоставить пароль.
- Значение *current поддерживается только в системах версии V4R3 и выше. Если программа выполняется в системе V4R2, то в ней необходимо задать пароль и ИД пользователя.

Примеры

The following examples show how to use the AS400 object with the i5/OS JVM.

Example: Creating an AS400 object when the i5/OS JVM is running a Java program

When a Java program is running in the i5/OS JVM, the program does not need to supply a system name, user ID, or password.

Примечание: При обращении к объектам на уровне записей пользователь **должен** предоставить пароль.

Если указанные значения не будут заданы, объект AS400 подключится к локальной системе, указав ИД пользователя и пароль задания, запустившего программу на Java.

When the program is running on the i5/OS JVM, setting the system name to **localhost** is the same as not setting the system name. Ниже приведен пример подключения к текущему серверу:

```
     // Создание двух объектов AS400. Если программа на Java выполняется
// в JVM для i5/OS, объекты выполняют одинаковую функцию.
     // Они создают соединение с текущим сервером с помощью ИД пользователя и
     // пароля задания, запустившего программу на Java.
     AS400 sys  = new AS400()
     AS400 sys2 = new AS400("localhost")
```

Example: Connecting to the current server with a different user ID and password from the program that started the job The Java program can set a user ID and password even when the program is running on the i5/OS JVM. Эти значения переопределят ИД пользователя и пароль задания, запустившего программу на Java.

В приведенном ниже примере программа на Java подключается к текущему серверу, указывая ИД пользователя и пароль, отличные от тех, которые определены в задании, запустившем программу на Java.

```
     // Создание объекта AS400. Программа подключится к текущему серверу,
// не используя ИД пользователя и пароль задания, запустившего
     // в программе на Java. Применяются указанные значения.
     AS400 sys = new AS400("localhost", "USR2", "PSWRD2")
```

Example: Connecting to a different server by using the user ID and password of the job that started the Java program

R A Java program that is running on one server can connect to and use the resources of other systems.

Если в качестве ИД пользователя и пароля будет задано значение ***current**, то при подключении к целевой системе будут применяться ИД пользователя и пароль задания, запустившего программу на Java, а не самой программы на Java.

В приведенном ниже примере программа на Java, запущенная на одном сервере, использует ресурсы другого сервера. При подключении к целевому серверу будут применяться ИД пользователя и пароль задания, запустившего программу на Java.

```
     // Создание объекта AS400. Программа, запущенная на одном сервере,
// установит соединение с другим сервером (называемым "целевым").
     // Так как в качестве ИД пользователя и пароля задано
     // специальное значение *current, при
```

```
// подключении к целевому серверу применяются ИД пользователя
// и пароль задания, запустившего программу на Java.
AS400 target = new AS400("target", "*current", "*current")
```

Обзор свойств объекта AS400, связанных с идентификацией пользователей:

В следующей таблице перечислены особенности обработки ИД пользователя и пароля программой на Java на сервере и программой на Java на компьютерах-клиентах:

Значения объекта AS400	Программа на Java на сервере	Программа на Java на клиенте
Имя системы, ИД пользователя и пароль не заданы	Подключается к текущему серверу с ИД пользователя и паролем задания, запустившего программу	Запрашивает имя системы, ИД пользователя и пароль
Имя системы = localhost	Подключается к текущему серверу с ИД пользователя и паролем задания, запустившего программу	**Ошибка:** Значение localhost недопустимо, когда программа на Java выполняется на клиенте
Имя системы = ИД пользователя локальной системы = *current		
Имя системы = ИД пользователя локальной системы = *current Пароль = *current		
Имя системы = ″sys″	Подключение к системе ″sys″ с ИД и паролем задания, из которого запущена программа. ″sys″ - текущий или удаленный сервер.	Запрашивает ИД пользователя и пароль
Имя системы = ИД пользователя локальной системы = ″UID″ Пароль = ″PWD″	Подключается к текущему серверу с указанными ИД пользователя и паролем программы на Java, а не задания, запустившего программу	**Ошибка:** Значение localhost недопустимо, когда программа на Java выполняется на клиенте

Независимый пул вспомогательной памяти (ASP)

Независимый пул вспомогательной памяти (IASP) - это набор дисков, который можно включить или выключить независимо от остальной памяти системы.

Независимые ASP могут содержать:

- пользовательские файловые системы
- внешние библиотеки

Любой IASP содержит всю необходимую системную информацию о хранящихся в нем данных. Поэтому IASP можно включить, выключить или перенести в другую систему во время работы системы.

Дополнительная информация приведена в разделах Независимые ASP и Пользовательские ASP.

ASP, к которому необходимо подключиться, можно задать с помощью ″имя базы данных″ свойства JDBC или метода setDatabaseName() method класса AS400JDBCDataSource.

All other IBM Toolbox for Java classes (IFSFile, Print, DataQueues, and so on) use the Independent ASP specified by the job description of the user profile that connects to the server.

Исключительные ситуации

The IBM Toolbox for Java access classes throw exceptions when device errors, physical limitations, programming errors, or user input errors occur. Исключительные ситуации разделены на классы в соответствии с характером ошибок, а не по месту их возникновения.

При возникновении большинства исключительных ситуаций передается следующая информация:

- **Тип ошибки:** Объект исключительной ситуации указывает тип ошибки. Ошибки одного типа объединены в класс исключительных ситуаций.
- **Сведения об ошибке:** Объект исключительной ситуации содержит код возврата, указывающий на причину возникновения ошибки. Коды возврата являются константами, определенными в классе исключительных ситуаций.
- **Текст ошибки:** Объект исключительной ситуации содержит строку - описание ошибки. Эта строка переведена на язык, определяемый локалью виртуальной машины Java клиента.

Пример: Обработка исключительной ситуации

В следующем примере перехватывается исключительная ситуация, считывается код возврата и выводится текст об ошибке:

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
// Все предварительные операции по удалению файла на сервере с помощью класса IFSFile
// выполнены. Попытайтесь удалить файл.
try
{
   aFile.delete();

}

// При удалении возникла ошибка.
catch (ExtendedIOException e)
{
   // Показать преобразованную строку, содержащую описание причины сбоя
   // при удалении.
   System.out.println(e);

   // Получение кода возврата объекта исключительной ситуации и просмотр
   // дополнительной информации для этого кода возврата.
   int rc = e.getReturnCode()

   switch (rc)
   {
      case ExtendedIOException.FILE_IN_USE:
         System.out.println("Удаление невозможно - файл используется"):
         break;

      case ExtendedIOException.PATH_NOT_FOUND:
         System.out.println("Удаление невозможно - путь не найден"):
         break;

      // Для каждой конкретной ошибки, которую необходимо отследить...

      default:
         System.out.println("Удаление невозможно - rc = ");
         System.out.println(rc);
   }
}
```

События при ошибках

In most cases, the IBM Toolbox for Java GUI components fire error events instead of throw exceptions.

Событие при ошибке является внешним представлением исключительной ситуации, вызванной встроенным компонентом.

Вы можете создать обработчик событий для всех ошибок, о которых может сообщить определенный компонент GUI. При появлении исключительной ситуации обработчик будет вызываться для выполнения требуемой обработки. События при ошибках, напротив, по умолчанию игнорируются.

The IBM Toolbox for Java provides a graphical user interface component called ErrorDialogAdapter, which automatically displays a dialog to the user whenever an error event is fired.

Примеры

Ниже приведены примеры обработки ошибок и определения простого обработчика ошибок.

Пример: Обработка событий при ошибках с помощью окна диалога

Следующий пример иллюстрирует обработку события при возникновении ошибки с выводом окна диалога:

```
// Все компоненты графического интерфейса пользователя
// созданы и размечены. Теперь к компонентам нужно добавить обработчик ErrorDialogAdapter.
// Этот метод будет создавать сообщения об ошибках компонента с помощью
// окна диалога.

ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (parentFrame);
component.addErrorListener (errorHandler);
```

Пример: Определение обработчика ошибок

Вы можете создать обработчик событий по-другому. Для этого можно воспользоваться интерфейсом ErrorListener.

Ниже показан пример простого обработчика ошибок, который заносит сообщение об ошибке в System.out:

```
class MyErrorHandler
implements ErrorListener
{
   // Этот метод вызывается при возникновении ошибки.
   public void errorOccurred(ErrorEvent event)
   {
       Exception e = event.getException ();
       System.out.println ("Ошибка: " + e.getMessage ());
   }
}
```

Пример: Обработка событий при ошибках с помощью обработчика ошибок

В следующем примере продемонстрировано применение обработчика ошибок для компонента графического интерфейса:

```
MyErrorHandler errorHandler = new MyErrorHandler ();
component.addErrorListener (errorHandler);
```

Ссылки, связанные с данной

“Классы Vaccess” на стр. 251
Пакет Vaccess и его классы устарели. Рекомендуется использовать пакет Access в сочетании с Java Swing.

“Исключительные ситуации” на стр. 47
The IBM Toolbox for Java access classes throw exceptions when device errors, physical limitations, programming errors, or user input errors occur. Исключительные ситуации разделены на классы в соответствии с характером ошибок, а не по месту их возникновения.

Информация, связанная с данной

ErrorDialogAdapter Javadoc

Класс Trace

Класс Trace позволяет заносить в протокол точки трассировки и диагностические сообщения в программах на Java. Эта информация помогает воспроизводить неполадки и выполнять их диагностику.

Примечание: Можно также задать трассировку с помощью системных свойств трассировки.

Класс Trace регистрирует данные следующих категорий:

Категория данных	Описание
Преобразование	Регистрирует преобразование символов из кодовой страницы в формат Unicode и наоборот. Эта категория применяется только классами IBM Toolbox for Java.
Поток данных	Logs the data that flows between the system and the Java program. Эта категория применяется только классами IBM Toolbox for Java.
Диагностика	Регистрирует информацию о состоянии.
Ошибка	Регистрирует дополнительные ошибки, вызвавшие исключительную ситуацию.
Информация	Отслеживает поток данных в программе.
PCML	Эта категория применяется для определения способа интерпретации данных PCML, отправляемых на сервер и получаемых с сервера.
Сервер Proxy	Эта категория применяется классами IBM Toolbox for Java для сбора информации о данных, которыми обмениваются клиент и сервер Proxy.
Предупреждение	Регистрирует информацию об исправимых ошибках программы.
Все	Эта категория позволяет разрешить или запретить трассировку всех вышеперечисленных категорий. Данные трассировки для этой категории нельзя занести в протокол напрямую.

Классы IBM Toolbox for Java также применяют категории трассировки. При включении трассировки в программе на Java информация IBM Toolbox for Java регистрируется вместе с информацией приложения.

Вы можете выполнить трассировку для одной или нескольких категорий.. Once the categories are selected, use the setTraceOn method to turn tracing on and off. Data is written to the log using the log method.

Данные трассировки различных компонентов могут направляться в разные протоколы. Обычно данные трассировки записываются в протокол по умолчанию. Для записи данных трассировки приложения в другой протокол или стандартный вывод применяется трассировка компонентов. С ее помощью можно отделить данные трассировки приложения от остальных данных.

Чрезмерное занесение информации в протокол может снизить производительность. Use the isTraceOn method to query the current state of the trace. С помощью этого метода программа на Java определяет, нужно ли перед вызовом метода log создавать запись трассировки. Вызов метода log в то время, когда ведение протокола отключено, не приводит к ошибке, но снижает производительность.

По умолчанию информация протокола записывается в стандартный вывод. To redirect the log to a file, call the setFileName() method from your Java application. В общем случае это возможно только для приложений Java, так как большинство браузеров не позволяют аплетам записывать информацию в локальную файловую систему.

По умолчанию ведение протокола отключено. В программах на Java должна быть предусмотрена возможность включить ведение протокола. Например, в приложении можно определить параметр командной строки, задающий категории заносимых в протокол данных. В этом случае пользователь сможет задать этот параметр, как только ему потребуется собрать некоторую информацию.

Примеры

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведены примеры использования класса Trace.

Пример: Применение метода setTraceOn() и запись данных в протокол с помощью метода log

```
     // Включение занесения диагностики, информации и предупреждений в протокол.
     Trace.setTraceDiagnosticOn(true);
     Trace.setTraceInformationOn(true);
     Trace.setTraceWarningOn(true);

     // Включение трассировки.
     Trace.setTraceOn(true);

     // ... Запись информации в протокол.
     Trace.log(Trace.INFORMATION, "Вызов метода xxx класса xxx");

     // Отключение трассировки.
     Trace.setTraceOn(false);
```

Пример: Применение класса Trace

В следующем примере второй способ применения класса Trace является более предпочтительным.

```
     // Способ 1 - создание записи трассировки,
     // вызов метода log и определение с помощью класса трассировки
     // нужно ли записывать данные. Такой способ работает медленнее,
// чем приведенный ниже.
     String traceData = new String("Вход в класс xxx, данные = ");
     traceData = traceData + data + "состояние = " + state;
     Trace.log(Trace.INFORMATION, traceData);

     // Способ 2 - проверка состояния протокола перед созданием
     // записи. Этот метод эффективнее при отсутствии трассировки.
if (Trace.isTraceOn() && Trace.isTraceInformationOn())
     {
        String traceData = new String("Вход в класс xxx, данные = ");
        traceData = traceData + data + "состояние = " + state;
        Trace.log(Trace.INFORMATION, traceData);
     }
```

Пример: Трассировка отдельных компонентов

```
     // Создание строки с названием компонента. Создание объекта
// эффективнее, чем применение нескольких строковых литералов.
     String myComponent1 = "com.myCompany.xyzComponent";
     String myComponent2 = "com.myCompany.abcComponent";

     // Сохранение данных общей трассировки и трассировки отдельных компонентов в разных файлах.
     // Файл общей трассировки будет содержать всю информацию, а файл трассировки
     // отдельного компонента - только информацию, относящуюся к этому
     // компоненту. Если файл трассировки не указан, все данные трассировки
// передаются в стандартный вывод с указанием компонента
     // перед каждым сообщением.

     // Trace.setFileName("c:\\bit.bucket");
     // Trace.setFileName(myComponent1, "c:\\Component1.log");
     // Trace.setFileName(myComponent2, "c:\\Component2.log");

     Trace.setTraceOn(true);             // Включение трассировки.
     Trace.setTraceInformationOn(true);  // Запись информационных сообщений.

     // Занесение в протокол трассировки данных для отдельных
```

```
// компонентов и общих данных трассировки.

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

Информация, связанная с данной

Trace Javadoc

Оптимизация i5/OS

Лицензионная программа IBM Toolbox for Java написана на языке Java, поэтому она может работать на любой платформе, для которой предусмотрена виртуальная машина Java (JVM). Классы IBM Toolbox for Java также могут применяться на любой платформе.

Additional classes come with i5/OS that enhance the behavior of the IBM Toolbox for Java when it is running on the i5/OS JVM. Sign-on behavior and performance are improved when running on the i5/OS JVM and connecting to the same server. i5/OS incorporated the additional classes starting at Version 4 Release 3.

Включение функции оптимизации

IBM Toolbox for Java поставляется в виде двух пакетов: в качестве отдельной лицензионной программы и вместе с i5/OS.

- Licensed Program 5761-JC1. Файлы лицензионной программы IBM Toolbox for Java расположены в следующем каталоге:

  ```
  /QIBM/ProdData/http/public/jt400/lib
  ```

  These files do not contain i5/OS optimizations. Эти файлы IBM Toolbox for Java следует применять в том случае, если программа на сервере и на клиенте должна выполняться с одинаковой скоростью.
- i5/OS. IBM Toolbox for Java is also shipped with i5/OS in directory

  ```
  /QIBM/ProdData/OS400/jt400/lib
  ```

  These files do contain the classes that optimize the IBM Toolbox for Java when running on the i5/OS JVM.

  Дополнительная информация приведена в Примечании 1 к разделу Файлы Jar.

Соглашения о входе в систему

With the additional classes provided with i5/OS, Java programs have additional options for providing server (system) name, user ID and password information to the IBM Toolbox for Java.

When accessing a System i resource, the IBM Toolbox for Java classes must have a system name, user ID and password.

- **При работе в клиентской системе** имя системы, ИД пользователя и пароль должны быть предоставлены программой на Java. Если они не заданы в программе, то эти значения будут запрошены IBM Toolbox for Java у пользователя при входе в систему.
- **When running on the i5/OS Java virtual machine**, the IBM Toolbox for Java has one more option. В этом случае программа на Java может отправлять запросы локальному серверу, указывая ИД пользователя и пароль запустившего ее задания.

With the additional classes, the user ID and password of the current job also can be used when a Java program that is running on one System i5 accesses the resources on another System i5. В этом случае программа на Java должна задать имя системы и указать специальное значение ″*current″ вместо ИД пользователя и пароля.

При обращении к отдельным записям файла программа на Java может указать пароль ″*current″ только в версии V4R4 или выше. В других версиях при обращении к файлу на уровне записей можно задать имя системы ″localhost″ и ИД пользователя ″*current″. Тем не менее, программа на Java должна самостоятельно предоставить пароль.

Программа на Java задает имя системы, ИД пользователя и пароль в объекте AS400.

Для того чтобы применялись ИД пользователя и пароль задания, программа на Java должна указать в качестве ИД пользователя и пароля значение ″*current″, либо вызвать конструктор, в котором нет параметров ИД пользователя и пароля.

Для работы с локальной системой iSeries программа на Java должна указать имя системы ″localhost″ или вызвать конструктор по умолчанию. Это означает, что

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

равносильно

```
AS400 system = new AS400("localhost", "*current", "*current");
```

Примеры

Ниже приведены примеры входа на сервер с помощью оптимизированных классов.

Пример: Вход в систему с применением различных конструкторов AS400

В следующем примере создаются два объекта AS400. Они выполняют одинаковые функции: запускают команды на текущем сервере с помощью ИД пользователя и пароля задания. При создании первого объекта в качестве ИД пользователя и пароля указывается специальное значение, а при создании второго вызывается конструктор по умолчанию без параметров.

```
R                       // Создание объекта AS400. Используется конструктор
R                       // по умолчанию, в котором не задается имя системы,
R                       // ИД пользователя и пароль. Следовательно, объект AS400
R                       // requests to the local server using the job's
R                       // с помощью ИД пользователя и пароля задания. Если бы
R // программа запускалась на клиенте, то появлялось бы
R                       // приглашение на ввод имени системы, ИД пользователя и пароля.
R     AS400 sys1 = new AS400();
R
R                       // Создание объекта AS400. Объект отправляет
R                       // requests to the local System i5 using the job's
R                       // ИД пользователя и пароля задания. Такой вариант
R // инициализации неприменим в клиентской системе.
R     AS400 sys2 = new AS400("localhost", "*current", "*current");
R
R                       // Создание двух объектов вызова команд, работающих
R                       // с ранее созданными объектами AS400.
R     CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
R     CommandCall cmd2 = new CommandCall(sys2,"myCommand2");
R
R                       // Запуск команд.
R     cmd1.run();
R     cmd2.run();
```

Пример: Вход в систему с помощью ИД пользователя и пароля текущего задания

R In the following example an AS400 object is created that represents a second System i. Since ″*current″ is used, the
R job’s user ID and password from the server running the Java program are used on the second (target) server.

```
                        // Создание объекта AS400. Объект отправляет
                        // requests to a second System i using the user ID
                        // и пароля задания текущего сервера.
     AS400 sys = new AS400("mySystem.myCompany.com", "*current", "*current");


                        // Создание объекта вызова команды для запуска команды
                        // на целевом сервере.
     CommandCall cmd = new CommandCall(sys,"myCommand1");
```

```
                    // Запуск команды.
    cmd.run();
```

Повышение производительности

With the additional classes provided by i5/OS, Java programs running on the Java virtual machine for i5/OS experience improved performance. Performance is improved in some cases because less communication function is used, and in other cases, an API is used instead of calling the server program.

Сокращение времени загрузки

Для того чтобы сократить до минимума число загружаемых файлов классов IBM Toolbox for Java, воспользуйтесь сервером Proxy и инструментом AS400ToolboxJarMaker.

Повышение скорости обмена данными

R For all IBM Toolbox for Java functions except JDBC and integrated file system access, Java programs running on the
R Java virtual machine for i5/OS will run faster. Это связано с тем, что виртуальная машина Java вызывает меньше
R функций связи для подключения программы к серверу.

R Функции для работы с JDBC и интегрированной файловой системы не были оптимизированы, так как
R соответствующие средства их оптимизации уже существуют. When running on System i, you can use the JDBC
R driver for i5/OS instead of the JDBC driver that comes with the IBM Toolbox for Java. Для работы с файлами
R сервера рекомендуется использовать пакет java.io вместо классов доступа к интегрированной файловой
R системе, входящих в состав IBM Toolbox for Java.

Прямой вызов API i5/OS

Следующие классы, входящие в состав IBM Toolbox for Java, обеспечивают высокую производительность, поскольку в них вместо вызова программы сервера напрямую вызываются API i5/OS.

- Классы AS400Certificate
- Класс CommandCall
- Класс DataQueue
- Класс ProgramCall
- Классы доступа к базе данных на уровне записей
- Класс ServiceProgramCall
- Класс UserSpace

Явный вызов API допустим только в том случае, если применяется ИД пользователя и пароль задания, запустившего программу на Java. Следовательно, для того чтобы воспользоваться этим средством повышения производительности, нужно задать ИД пользователя и пароль задания, запустившего программу на Java. Вместо имени системы рекомендуется указать значение ″localhost″, а вместо ИД пользователя и пароля - значение ″*current″.

Изменение правил назначения порта

В схему назначения порта внесены усовершенствования, ускорившие доступ к портам. Раньше запрос на назначение порта отправлялся специальной программе. From there, the server would determine which port was available and return that port to the user to be accepted. Теперь вы можете либо самостоятельно задать номер порта, либо указать, что должен применяться порт по умолчанию. При этом не затрачивается время на автоматический выбор порта. Для просмотра и изменения списка портов сервера предназначена команда WRKSRVTBLE.

Для поддержки новой схемы назначения портов в класс AS/400 было добавлено несколько новых методов:

- getServicePort
- setServicePort
- setServicePortsToDefault

Текстовые файлы для разных языков

Текстовые файлы для разных языков поставляются в составе программы IBM Toolbox for Java в виде файлов классов, а не файлов свойств. The server finds messages in class files faster than in property files. Метод ResourceBundle.getString() теперь также выполняется быстрее, так как обрабатываемые им файлы расположены в каталоге, который первым просматривается во время поиска. Кроме того, поиск переведенных сообщений теперь также выполняется быстрее.

Программы преобразования

R Two classes allow faster, more efficient conversion between Java and the system:

- Binary Converter: Converts between Java byte arrays and Java simple types.
- Character Converter: Converts between Java String objects and i5/OS code pages.

В пакет IBM Toolbox for Java теперь входят таблицы преобразования для более чем 100 наиболее распространенных CCSID. Ранее IBM Toolbox for Java выполнял почти все текстовые преобразования с помощью средств Java. Если в языке Java не было необходимой таблицы преобразования, IBM Toolbox for Java загружал ее с сервера.

IBM Toolbox for Java выполняет все текстовые преобразования для входящих в пакет CCSID. При обнаружении CCSID, не входящего в пакет, он пытается преобразовать данные с помощью средств языка Java. Таблицы преобразования больше не загружаются с сервера IBM Toolbox for Java. Такой подход позволил существенно сократить время, уходящее на преобразование текста в приложении IBM Toolbox for Java. Для применения новых способов преобразования никаких действий от пользователя не требуется; повышение производительности происходит на более низком уровне преобразования.

Информация, связанная с данной

AS400 Javadoc

BinaryConverter Javadoc

Character Converter Javadoc

Установка и обновление классов на клиенте

For most installation and update purposes, the IBM Toolbox for Java classes can be referenced at their location in the integrated file system on the server.

Так как к этому расположению применяются временные исправления программ (PTF), программы на Java , обращающиеся к классам непосредственно на сервере, получают обновления автоматически. But, accessing the classes from the server does not always work, specifically for the following situations:

- Если рабочая станция подключена к серверу по линии связи с низким быстродействием, то загрузка классов с сервера на рабочую станцию будет выполняться слишком медленно.
- R L If Java applications use the CLASSPATH environment variable to access the classes on the client file system, you need System i Access for Windows to redirect file system calls to the server. It may not be possible for System i Access for Windows to reside on the client.

В перечисленных выше случаях оптимальным вариантом является установка классов в клиентской системе.

AS400ToolboxJarMaker

Формат файлов JAR был специально разработан для ускорения загрузки файлов программ на Java. Класс AS400ToolboxJarMaker еще больше сокращает время загрузки путем сокращения размера файла JAR IBM Toolbox for Java.

Кроме этого, класс AS400ToolboxJarMaker позволяет распаковывать файлы JAR с целью получить доступ к их отдельным компонентам.

Гибкость класса AS400ToolboxJarMaker

Все функции работы с архивами JAR реализованы с помощью класса JarMaker и его подкласса AS400ToolboxJarMaker:

- Более общий инструмент JarMaker предназначен для работы с любыми файлами JAR и ZIP. Он позволяет разбивать архивы и сокращать их размер путем удаления ненужных классов.
- The AS400ToolboxJarMaker customizes and extends JarMaker functions for easier use with IBM Toolbox for Java JAR files.

По вашему усмотрению вы можете использовать методы AS400ToolboxJarMaker в программе на Java или вызывать их как независимые приложения из командной строки. Для вызова AS400ToolboxJarMaker из командной строки введите:

```
java utilities.JarMaker [опции]
```

где

- опции - одна или несколько опций

For a complete set of options available to run at a command line prompt, see the following in the Javadoc:

- Options for the JarMaker base class
- Extended options for the AS00ToolboxJarMaker subclass

Применение AS400ToolboxJarMaker

Класс AS400ToolboxJarMaker позволяет работать с файлами JAR несколькими способами:

- Распаковывать файлы, находящиеся в файлах JAR
- Разбивать большие файлы JAR на несколько файлов JAR меньшего размера
- Exclude any IBM Toolbox for Java files that your application does not need to run

Распаковка файла JAR

Предположим, что вы хотите распаковать один файл из архива JAR. Класс AS400ToolboxJarMaker позволяет получить файл из архива и поместить его в одно из следующих мест:

- Current directory (extract(jarFile))
- Another directory (extract(jarFile, outputDirectory))

Например, следующий фрагмент кода позволяет распаковать класс AS400.class и все зависящие от него классы из архива jt400.jar:

```
java utilities.AS400ToolboxJarMaker -source jt400.jar
    -extract outputDir
    -requiredFile com/ibm/as400/access/AS400.class
```

Разбиение одного файла JAR на несколько файлов JAR меньшего размера

Предположим, вы хотите разбить файл JAR на файлы меньшего размера, задав максимально допустимый размер файла JAR. AS400ToolboxJarMaker, accordingly, provides you with the split(jarFile, splitSize) function.

В следующем примере файл jt400.jar разбивается на файлы JAR размером не более 300 Кб:

```
java utilities.AS400ToolboxJarMaker -split 300
```

Удаление ненужных файлов из архива JAR

With AS400ToolboxJarMaker, you can exclude any IBM Toolbox for Java files not needed by your application by selecting only the IBM Toolbox for Java components, languages, and CCSIDs that you need to make your application run. Кроме того, AS400ToolboxJarMaker позволяет добавлять и удалять файлы JavaBean, связанные с выбранными компонентами.

For example, the following command creates a JAR file that contains only those IBM Toolbox for Java classes needed to make the CommandCall and ProgramCall components of the IBM Toolbox for Java work:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall
```

Кроме этого, если преобразование строк между кодировкой Unicode и набором двухбайтовых символов (DBCS) не требуется, вы можете сократить размер файла JAR на 400 Кб, опустив ненужные таблицы преобразования с помощью опции -ccsid:

```
java utilities.AS400ToolboxJarMaker -component CommandCall,ProgramCall -ccsid 61952
```

Примечание: Классы преобразования не входят в состав классов вызова программ. Если в файл JAR добавляются классы вызова программ, классы преобразования должны быть указаны явно с помощью опции -ccsid.

JarMaker Javadoc

AS400ToolboxJarMaker Javadoc

Java national language support

Java поддерживает не все национальные языки, предусмотренные на сервере.

When a mismatch between languages occurs, for example, if you are running on a local workstation that is using a language that is not supported by Java, the IBM Toolbox for Java licensed program **may issue some error messages in English**.

Обслуживание и поддержка IBM Toolbox for Java

Ниже перечислены различные источники информации и средства, применяемые для обслуживания и поддержки.

IBM Toolbox for Java troubleshooting information Use this information to help you resolve problems when using IBM Toolbox for Java.

JTOpen/IBM Toolbox for Java forum Join the community of Java programmers who use IBM Toolbox for Java. На этом форуме вам с удовольствием окажут помощь программисты на Java, а может быть - даже сами разработчики IBM Toolbox for Java.

R **Server support** Use the IBM Server support Web site to find out about the tools and resources that help
R you streamline the technical planning and support for your system.

Поддержка программного обеспечения Web-сайт IBM Software Support Services содержит информацию о самых разных службах поддержки программного обеспечения IBM.

R Support services for the IBM Toolbox for Java, 5761-JC1, are provided under the usual terms and conditions for
R software products. Услуги по поддержке включают программные службы, консультации специалистов и
R справочные службы. За дополнительной информацией обратитесь в представительство фирмы IBM.

Программные службы и специалисты отвечают за исправление ошибок в программе IBM Toolbox for Java, а справочная служба предназначена для решения вопросов, связанных с прикладными программами и их отладкой.

Справочная служба не отвечает на следующие вопросы, связанные с API IBM Toolbox for Java, в следующих случаях:

- Вопросы, связанные с ошибками API Java, если их можно воспроизвести, создав простой тестовый пример.
- Вопросы, связанные с пояснением документации
- Вопросы о том, где можно найти примеры кода и документацию

Справочная служба отвечает на вопросы о любых программах, включая примеры программ, поставляемые вместе с лицензионной программой IBM Toolbox for Java. Additional samples may be made available on the Internet at the System i home page  on an unsupported basis.

Информация об устранении неполадок поставляется вместе с лицензионной программой IBM Toolbox for Java. Если вы считаете, что в API IBM Toolbox for Java есть ошибка, вам потребуется создать простой пример, моделирующий ситуацию, в которой возникает эта ошибка.

Примеры программ

The following list provides links to entry points for many of the examples used throughout the IBM Toolbox for Java information.

Классы доступа	Компоненты JavaBean	Классы трассировки соединений
Graphical Toolbox	Классы HTML	PCML
Классы составителя отчетов	Классы ресурсов	RFML
Классы защиты	Классы сервлетов	Простые примеры
Советы программисту	“Examples: ToolboxME” на стр. 703	Классы утилит
Классы Vaccess	Классы XPCML	

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

ЗА ИСКЛЮЧЕНИЕМ УСТАНОВЛЕННЫХ ЗАКОНОМ ГАРАНТИЙ, ОТКАЗ ОТ КОТОРЫХ НЕВОЗМОЖЕН, ФИРМА IBM И РАЗРАБОТЧИКИ И ПОСТАВЩИКИ ЕЕ ПРОГРАММ НЕ ДАЮТ НИКАКИХ ГАРАНТИЙ И ОБЯЗАТЕЛЬСТВ, НИ ЯВНЫХ, НИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ЭТИМ, ПОДРАЗУМЕВАЕМЫЕ ГАРАНТИИ И ОБЯЗАТЕЛЬСТВА ОТНОСИТЕЛЬНО КОММЕРЧЕСКОЙ ЦЕННОСТИ, ПРИГОДНОСТИ ДЛЯ КАКОЙ-ЛИБО КОНКРЕТНОЙ ЦЕЛИ И СОБЛЮДЕНИЯ АВТОРСКИХ ПРАВ, ПО ОТНОШЕНИЮ К ПРОГРАММАМ И ТЕХНИЧЕСКОЙ ПОДДЕРЖКЕ, ЕСЛИ ТАКОВЫЕ ПРЕДОСТАВЛЯЮТСЯ.

НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ ФИРМА IBM И РАЗРАБОТЧИКИ И ПОСТАВЩИКИ ЕЕ ПРОГРАММ НЕ НЕСУТ ОТВЕТСТВЕННОСТЬ НИ ЗА КАКОЕ ИЗ СЛЕДУЮЩИХ СОБЫТИЙ, ДАЖЕ ЕСЛИ ОНИ БЫЛИ ЗАРАНЕЕ ИНФОРМИРОВАНЫ О ВОЗМОЖНОСТИ НАСТУПЛЕНИЯ ЭТИХ СОБЫТИЙ:

1. ПОТЕРЯ ИЛИ ПОВРЕЖДЕНИЕ ДАННЫХ;
2. ПРЯМЫЕ, СПЕЦИАЛЬНЫЕ, СЛУЧАЙНЫЕ ИЛИ КОСВЕННЫЕ УБЫТКИ, ЛИБО ЛЮБЫЕ ВЗАИМОСВЯЗАННЫЕ УБЫТКИ; ИЛИ
3. НЕПОЛУЧЕННЫЕ ПРИБЫЛЬ, ВЫГОДА, ДОХОД, ПРЕСТИЖ ИЛИ ПРЕДПОЛАГАЕМАЯ ЭКОНОМИЯ СРЕДСТВ.

В ЗАКОНОДАТЕЛЬСТВАХ НЕКОТОРЫХ СТРАН НЕ ДОПУСКАЕТСЯ ОТКАЗ ИЛИ ОГРАНИЧЕНИЕ ОТВЕТСТВЕННОСТИ ЗА ПРЯМЫЕ СЛУЧАЙНЫЕ ИЛИ ВЗАИМОСВЯЗАННЫЕ УБЫТКИ, ПОЭТОМУ НЕКОТОРЫЕ ИЛИ ВСЕ УКАЗАННЫЕ ВЫШЕ ОГРАНИЧЕНИЯ И ОГОВОРКИ МОГУТ НЕ ИМЕТЬ СИЛЫ В ВАШЕМ СЛУЧАЕ.

Примеры: Классы доступа

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java access classes.

AS400JPing

- Example: Using AS400JPing within a Java program

BidiTransform

- Пример: Преобразование двунаправленного текста с помощью класса AS400BidiTransform

Класс CommandCall

- Пример: Запуск команды на сервере с помощью класса CommandCall
- Пример: Применение класса CommandCall для получения имени сервера и команды и последующего вывода результатов

ConnectionPool

- Пример: Создание соединений с сервером с помощью класса AS400ConnectionPool

DataArea

- Пример: Создание и использование области десятичных данных

Преобразование и описание данных

- Пример: Применение классов FieldDescription, RecordFormat и Record
- Пример: Помещение данных в очередь
- Пример: Получение данных из очереди
- Пример: Использование классов AS400DataType с ProgramCall

Класс DataQueue

- Пример: Создание объекта DataQueue, чтение данных и отключение
- Пример: Помещение данных в очередь
- Пример: Получение данных из очереди
- Пример: Занесение элементов в очередь с помощью класса KeyedDataQueue
- Пример: Извлечение объектов из очереди с помощью класса KeyedDataQueue

Цифровые сертификаты

- Пример: Получение списка цифровых сертификатов, принадлежащих пользователю

EnvironmentVariable

- Пример: Создание, настройка и получение значений переменных среды

Исключительные ситуации

- Пример: Обработка созданной исключительной ситуации, чтение кода возврата и вывод текста исключительной ситуации

FTP

- Пример: Копирование набора файлов из каталога сервера с помощью класса FTP
- Пример: Копирование набора файлов из каталога сервера с помощью подкласса AS400FTP

Интегрированная файловая система

- Примеры: Применение классов IFSFile
- Пример: Получение списка объектов каталога с помощью метода IFSFile.listFiles()
- Пример: Копирование файлов с помощью классов IFSFile
- Пример: Получение списка объектов каталога с помощью класса IFSFile
- Пример: Применение IFSJavaFile вместо java.io.File
- Пример: Получение списка объектов каталога сервера с помощью классов IFSFile

JavaApplicationCall

- Пример: Запуск из клиентской системы программы сервера, выводящей текст ″Hello World!″

JDBC

- Пример: Применение драйвера JDBC для создания и заполнения таблицы
- Пример: Применение драйвера JDBC для обработки запроса к таблице и вывода ее содержимого

Задания

- Пример: Получение и изменение информации о задании с помощью кэша
- Пример: Получение списка активных заданий
- Пример: Вывод сообщений из протокола задания, относящихся к определенному пользователю
- Пример: Получение идентификационной информации о задании указанного пользователя
- Пример: Получение списка заданий сервера с последующим выводом идентификаторов и информации о состоянии заданий
- Пример: Вывод сообщений из протокола задания текущего пользователя

Очереди сообщений

- Пример: Использование объекта очереди сообщений
- Пример: Вывод содержимого очереди сообщений
- Пример: Получение и печать сообщений
- Пример: Определение содержимого очереди сообщений
- Пример: Применение класса AS400Message совместно с CommandCall
- Пример: Применение класса AS400Message совместно с ProgramCall

NetServer

- Пример: Применение объекта NetServer для изменение имени NetServer

Печать

- Пример: Асинхронное получение списка буферных файлов с помощью интерфейса PrintObjectListListener
- Пример: Асинхронное получение списка буферных файлов *без* помощи интерфейса PrintObjectListListener
- Пример: Копирование буферного файла с помощью метода SpooledFile.copy()
- Пример: Создание буферного файла из потока ввода
- Пример: Создание потока данных SCS с помощью класса SCS3812Writer
- Пример: Чтение буферного файла
- Пример: Считывание и преобразование буферных файлов
- Пример: Синхронное получение списка буферных файлов

Права доступа

- Пример: Настройка прав доступа объекта AS400

Вызов программ

- Пример: Применение класса ProgramCall
- Пример: Запрос состояния программы с помощью класса ProgramCall
- Пример: Передача параметров с помощью объекта параметра программы

QSYSObjectPathName

- Пример: Построение имени файла интегрированной файловой системы
- Пример: Применение функции QSYSObjectPathName.toPath() для создания имени объекта AS400
- Пример: Применение класса QSYSObjectPathName для синтаксического анализа пути интегрированной файловой системы

Доступ на уровне записей

- Пример: Последовательный доступ к файлу
- Пример: Применение классов доступа на уровне записей для чтения файла
- Пример: Применение классов доступа на уровне записей для получение записей по ключу
- Пример: Применение класса LineDataRecordWriter

Вызовы служебных программ

- Пример: Вызов процедуры с помощью ServiceProgramCall

SystemStatus

- Пример: Использование кэширования с классом SystemStatus

Класс SystemPool

- Пример: Настройка максимального числа ошибок для SystemPool

SystemValue

- Пример: Применение классов SystemValue и SystemValueList

Класс Trace

- Пример: Применение метода Trace.setTraceOn()
- Пример: Рекомендуемый способ применения трассировки
- Пример: Трассировка отдельных компонентов

Класс UserGroup

- Пример: Получение списка пользователей
- Пример: Получение списка пользователей в группе

Класс UserSpace

- Пример: Создание пользовательского пространства

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: применение объекта CommandCall

This IBM Toolbox for Java example program prompts the user for the name of the server and the command to run, then prints the result of the command.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта CommandCall. Программа предлагает пользователю
// ввести имя сервера и запускаемую команду, после чего выдает результат
// выполнения команды.
//
// This source is an example of IBM Toolbox for Java "CommandCall"
//
////////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class CommandCallExample extends Object
{
   public static void main(String[] parmeters)
   {

      // Создание программы чтения ввода пользователя
            BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);



      // Определение переменных для имени системы и запускаемой команды
      String systemString  = null;
      String commandString = null;

      System.out.println( " " );



      // Получение от пользователя имени системы и запускаемой команды
      try
      {
         System.out.print("Имя системы: ");
         systemString = inputStream.readLine();

         System.out.print("Команда: ");
         commandString = inputStream.readLine();
      }
      catch (Exception e) {};

      System.out.println( " " );



      // Создание объекта AS400. Объект создается для целевой системы.
```

```
      AS400 as400 = new AS400(systemString);



      // Создание объекта вызова команд с указанием целевой системы.
CommandCall command = new CommandCall( as400 );



      try
      {
         // Запуск команды.
         if (command.run(commandString))
             System.out.print( "Команда выполнена успешно" );
         else
             System.out.print( "Команда не выполнена" );



         // Вывод сообщений команды.
         AS400Message[] messagelist = command.getMessageList();

         if (messagelist.length > 0)
         {
             System.out.println( ", сообщения команды:" );
             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( "Команда " + command.getCommand() + " не была запущена" );
      }

      System.exit(0);
   }
}
```

Пример: Применение AS400ConnectionPool

This IBM Toolbox for Java example program uses an AS400ConnectionPool to create connections to a system.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////////
//
// Пример работы с классом AS400ConnectionPooling. Эта программа использует
// create connections to a System i5.
// Формат вызова:
//    AS400ConnectionPooling система пользователь пароль
//
// Пример:
//    AS400ConnectionPooling MySystem MyUserId MyPassword
//
////////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class AS400ConnectionPooling
{
   public static void main (String[] parameters)
```

```
   {
      // Проверка входных параметров.
      if (parameters.length != 3)
      {
         System.out.println("");
         System.out.println("Формат:");
         System.out.println("");
         System.out.println("   AS400ConnectionPooling система пользователь пароль");
         System.out.println("");
         System.out.println("");
         System.out.println("Например:");
         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
      {
         // Создание пула AS400ConnectionPool.
         AS400ConnectionPool testPool = new AS400ConnectionPool();

         // Установка максимального количества соединений, равного 128.
         testPool.setMaxConnections(128);

         // Установка максимального срока действия, равного 30 минутам.
         testPool.setMaxLifetime(1000*60*30);     // Срок действия - 30 минут после создания

         // Создание 5 соединений, заранее подключенных к службе AS400.COMMAND.
         testPool.fill(system, userId, password, AS400.COMMAND, 1);
         System.out.println ();
                  System.out.println("К службе AS400.COMMAND подключено одно соединение");

         // Вызов getActiveConnectionCount и getAvailableConnectionCount для получения
         // количества используемых и доступных соединений с конкретной системой.
         System.out.println("Число активных соединений: " +
            testPool.getActiveConnectionCount(system, userId));
System.out.println("Число доступных соединений: " +
            testPool.getAvailableConnectionCount(system, userId));
// Создание соединения с службой AS400.COMMAND. (Необходимо использовать ограничения
         // на номер службы, определенные в классе AS400 (FILE, PRINT, COMMAND, DATAQUEUE и т.д.))
         // Поскольку параметры соединений уже заданы, времени на подключение
         // к службе не требуется.
         AS400 newConn1 = testPool.getConnection(system, userId, password, AS400.COMMAND);

         System.out.println ();
                  System.out.println("getConnection вернет заранее созданное соединение");
         System.out.println("Число активных соединений: " +
            testPool.getActiveConnectionCount(system, userId));
System.out.println("Число доступных соединений: " +
            testPool.getAvailableConnectionCount(system, userId));
// Создание нового объекта вызова команды и запуск команды
         CommandCall cmd1 = new CommandCall(newConn1);
         cmd1.run("CRTLIB FRED");

         // Возврат соединения в пул.
         testPool.returnConnectionToPool(newConn1);

         System.out.println ();
                  System.out.println("Соединение возвращено в пул");
         System.out.println("Число активных соединений: " +
            testPool.getActiveConnectionCount(system, userId));
```

```
System.out.println("Число доступных соединений: " +
            testPool.getAvailableConnectionCount(system, userId));
// Создание соединения с службой AS400.COMMAND. При этом будет возвращено то же
         // соединение, что и ранее.
         AS400 newConn2 = testPool.getConnection(system, userId, password, AS400.COMMAND);

         System.out.println ();
                  System.out.println("getConnection вернет заранее созданное соединение");
         System.out.println("Число активных соединений: " +
            testPool.getActiveConnectionCount(system, userId));
System.out.println("Число доступных соединений: " +
            testPool.getAvailableConnectionCount(system, userId));
// Создание соединения с службой AS400.COMMAND. При этом будет создано новое
         // соединение, поскольку предыдущее еще занято.
         AS400 newConn3 = testPool.getConnection(system, userId, password, AS400.COMMAND);

         System.out.println ();
                  System.out.println("getConnection создает новое соединение, поскольку доступных
                              соединений нет");
         System.out.println("Число активных соединений: " +
            testPool.getActiveConnectionCount(system, userId));
System.out.println("Число доступных соединений: " +
            testPool.getAvailableConnectionCount(system, userId));
// Закрытие тестового пула.
         testPool.close();
      }
      catch (Exception e)
      {
         // Если в любой из операций происходит сбой - выдается сообщение о сбое операции
         // с пулом и текст исключительной ситуации.

         System.out.println("Сбой операции с пулом");
         System.out.println(e);
         e.printStackTrace();
      }
   }
}
```

+ Examples: Using AS400JDBCManagedConnectionPoolDataSource class

+ These examples demonstrate the use of the AS400JDBCManagedConnectionPoolDataSource class. The
+ AS400JDBCManagedConnectionPoolDataSource class simplifies connection pool maintenance by eliminating the
+ need for user applications to implement their own management code.

+ **Примечание:** By using the code examples, you agree to the terms of the “Лицензия на исходный код и отказ от обязательств” на стр. 768.

+ Пример 1

+ This brief example shows the basic use of the AS400JDBCManagedConnectionPoolDataSource class.

```
+ import javax.naming.Context;
+ import javax.naming.InitialContext;
+ import javax.sql.DataSource;
+
+ import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
+ import com.ibm.as400.access.AS400JDBCManagedDataSource;
+
+
+ public class TestJDBCConnPoolSnippet
+ {
+
+     /**
+      * @param args
+      */
+     void test()
+     {
+         AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();
+
```

```
+         // Set general datasource properties.  Note that both connection pool datasource (CPDS) and managed
+                 // datasource (MDS) have these properties, and they might have different values.
+         cpds0.setServerName(host);
+         cpds0.setDatabaseName(host);//iasp can be here
+         cpds0.setUser(userid);
+         cpds0.setPassword(password);
+
+
+         cpds0.setSavePasswordWhenSerialized(true);
+
+         // Set connection pooling-specific properties.
+         cpds0.setInitialPoolSize(initialPoolSize_);
+         cpds0.setMinPoolSize(minPoolSize_);
+         cpds0.setMaxPoolSize(maxPoolSize_);
+         cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
+         cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
+         cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
+         //cpds0.setReuseConnections(false);  // do not re-use connections
+
+         // Set the initial context factory to use.
+
+         System.setProperty(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
+
+
+         // Get the JNDI Initial Context.
+         Context ctx = new InitialContext();
+
+         // Note: The following is an alternative way to set context properties locally:
+         //   Properties env = new Properties();
+         //   env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
+         //   Context ctx = new InitialContext(env);
+
+         ctx.rebind("mydatasource", cpds0);  // We can now do lookups on cpds, by the name "mydatasource".
+
+         // Create a standard DataSource object that references it.
+
+         AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
+         mds0.setDescription("DataSource supporting connection pooling");
+         mds0.setDataSourceName("mydatasource");
+         ctx.rebind("ConnectionPoolingDataSource", mds0);
+
+         DataSource dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
+
+         AS400JDBCManagedDataSource mds_ = (AS400JDBCManagedDataSource)dataSource_;
+
+         boolean isHealthy = mds_.checkPoolHealth(false);  //check pool health
+
+         Connection c = dataSource_.getConnection();
+
+     }
+
+ }
```

+ Пример 2

+ This example shows more details about how to use the AS400JDBCManagedConnectionPoolDataSource class.

```
+ import java.awt.TextArea;
+ import java.io.BufferedReader;
+ import java.io.File;
+ import java.io.FileReader;
+ import java.io.FileInputStream;
+ import java.io.FileOutputStream;
+ import java.io.OutputStream;
+ import java.io.PrintStream;
+ import java.util.Vector;
+ import java.util.Properties;
+
+ import java.sql.Connection;
+ import javax.sql.DataSource;
+ import java.sql.ResultSet;
+ import java.sql.Statement;
+ import javax.naming.*;
+ import java.util.Date;
+ import java.util.ArrayList;
+ import java.util.Random;
+ import com.ibm.as400.access.AS400;
+ import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
```

```
+ import com.ibm.as400.access.AS400JDBCManagedDataSource;
+ import com.ibm.as400.access.Trace;
+
+
+ public class TestJDBCConnPool
+ {
+
+     private static final boolean DEBUG = false;
+
+     // If you turn this flag on, be sure to also turn on the following flag:
+     //   AS400JDBCConnection.TESTING_THREAD_SAFETY.
+     private static final boolean TESTING_THREAD_SAFETY = false;
+
+     private static String userid;
+     private static String password;
+     private static String host;
+
+     // Note: For consistency, all time values are stored units of milliseconds.
+     private int initialPoolSize_;  // initial # of connections in pool
+     private int minPoolSize_;      // max # of connections in pool
+     private int maxPoolSize_;      // max # of connections in pool
+     private long maxLifetime_;     // max lifetime (msecs) of connections in pool
+     private long maxIdleTime_;     // max idle time (msecs) of available connections in pool
+     private long propertyCycle_;   // pool maintenance frequency (msecs)
+
+     private int numDaemons_;       // # of requester daemons to create
+     private static long timeToRunDaemons_; // total duration (msecs) to let the daemons run
+     private long daemonMaxSleepTime_;  // max time (msecs) for requester daemons to sleep each cycle
+     private long daemonMinSleepTime_;  // min time (msecs) for requester daemons to sleep each cycle
+     private long poolHealthCheckCycle_;  // # of msecs between calls to checkPoolHealth()
+
+     private boolean keepDaemonsAlive_ = true;  // When this is false, the daemons shut down.
+
+     private DataSource dataSource_;
+     private AS400JDBCManagedDataSource mds_;
+
+     private final Object daemonSleepLock_ = new Object();
+
+     private Random random_ = new Random();
+
+     static
+     {
+         try {
+             Class.forName("com.ibm.as400.access.AS400JDBCDriver");
+         }
+         catch (Exception e) {
+             System.out.println("Unable to register JDBC driver.");
+             System.exit(0);
+         }
+     }
+
+
+
+     public static void main(String[] args)
+     {
+         host = args[0];
+         userid = args[1];
+         password = args[2];
+         timeToRunDaemons_ = (new Integer(args[3])).intValue() * 1000; //milliseconds
+         //args[3]=time to run in seconds
+         TestJDBCConnPool cptest = new TestJDBCConnPool();
+
+         cptest.setup();
+         cptest.runTest();
+     }
+
+
+
+
+     public void setup()
+     {
+         try
+         {
+             if (DEBUG) System.out.println("TESTING_THREAD_SAFETY flag is " + (TESTING_THREAD_SAFETY
+ ? "true" : "false"));
+
+             if (TESTING_THREAD_SAFETY)
+             {
+                 // Adjust values for performing thread-intensive stress testing.
```

```
+                // NOTE: This assumes that the AS400JDBCConnection class has also been modified to
+                                    // not make actual connections to an actual server.
+                // To do this, edit AS400JDBCConnection.java, changing its TESTING_THREAD_SAFETY
+                                    // flag to 'false', and recompile.
+                minPoolSize_ = 100;
+                maxPoolSize_ = 190;
+                initialPoolSize_ = 150;  // this should get reset to maxPoolSize_
+                numDaemons_ = 75;
+                if (timeToRunDaemons_ == 0) {
+                    timeToRunDaemons_ = 180*1000; // 180 seconds == 3 minutes
+                }
+            }
+            else
+            {  // Set more conservative values, as we'll be making actual connections to an
+               // actual server, and we don't want to monopolize the server.
+                minPoolSize_ = 5;
+                maxPoolSize_ = 15;
+                initialPoolSize_ = 9;
+                numDaemons_ = 4;
+                if (timeToRunDaemons_ == 0) {
+                    timeToRunDaemons_ = 15*1000; // 15 seconds
+                }
+            }
+            maxLifetime_ = (int)timeToRunDaemons_ / 3;
+            maxIdleTime_ = (int)timeToRunDaemons_ / 4;
+            propertyCycle_ = timeToRunDaemons_ / 4;
+            poolHealthCheckCycle_ = Math.min(timeToRunDaemons_ / 4, 20*60*1000);
+            // at least once every 20 minutes (more frequently if shorter run-time)
+            daemonMaxSleepTime_ = Math.min(timeToRunDaemons_ / 3, 10*1000);
+            // at most 10 seconds (less if shorter run-time)
+            daemonMinSleepTime_ = 20;  // milliseconds
+
+            if (DEBUG) System.out.println("setup: Constructing
+ AS400JDBCManagedConnectionPoolDataSource (cpds0)");
+            AS400JDBCManagedConnectionPoolDataSource cpds0 = new
+ AS400JDBCManagedConnectionPoolDataSource();
+
+            // Set general datasource properties.  Note that both CPDS and MDS have these
+            // properties, and they might have different values.
+            cpds0.setServerName(host);
+            cpds0.setDatabaseName(host);//iasp can be here
+            cpds0.setUser(userid);
+            cpds0.setPassword(password);
+
+
+            cpds0.setSavePasswordWhenSerialized(true);
+
+            // Set connection pooling-specific properties.
+            cpds0.setInitialPoolSize(initialPoolSize_);
+            cpds0.setMinPoolSize(minPoolSize_);
+            cpds0.setMaxPoolSize(maxPoolSize_);
+            cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
+            cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
+            cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
+            //cpds0.setReuseConnections(false);  // don't re-use connections
+
+            // Set the initial context factory to use.
+
+
+            System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
+ "com.sun.jndi.fscontext.RefFSContextFactory");
+
+
+            // Get the JNDI Initial Context.
+            Context ctx = new InitialContext();
+
+            // Note: The following is an alternative way to set context properties locally:
+            //   Properties env = new Properties();
+            //   env.put(Context.INITIAL_CONTEXT_FACTORY,
+ "com.sun.jndi.fscontext.RefFSContextFactory");
+            //   Context ctx = new InitialContext(env);
+
+
+            ctx.rebind("mydatasource", cpds0);
+                        // We can now do lookups on cpds, by the name"mydatasource".
+
+            if (DEBUG) System.out.println("setup: lookup(\"mydatasource\"" + ")");
+          //   AS400JDBCManagedConnectionPoolDataSource cpds1 =
+ (AS400JDBCManagedConnectionPoolDataSource)ctx.lookup("mydatasource");
```

```
+            // if (DEBUG) System.out.println("setup: cpds1.getUser() == |" + cpds1.getUser() + "|");
+
+
+             // Create a standard DataSource object that references it.
+
+             if (DEBUG) System.out.println("setup: Constructing AS400JDBCManagedDataSource (mds0)");
+             AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
+             mds0.setDescription("DataSource supporting connection pooling");
+             mds0.setDataSourceName("mydatasource");
+             ctx.rebind("ConnectionPoolingDataSource", mds0);
+
+             if (DEBUG) System.out.println("setup: lookup(\"ConnectionPoolingDataSource\"" + ")");
+             dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
+             //dataSource_.setLogWriter(output_);
+             if (DEBUG) System.out.println("setup: dataSource_.getUser() == |" +
+ ((AS400JDBCManagedDataSource)dataSource_).getUser() + "|");
+
+             mds_ = (AS400JDBCManagedDataSource)dataSource_;
+
+
+         }
+         catch (Exception e)
+         {
+             e.printStackTrace();
+             System.out.println("Setup error during Trace file creation.");
+         }
+     }
+
+     void displayConnectionType(Connection conn, boolean specifiedDefaultId)
+     {
+         if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
+         {
+             System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "P)");
+         }
+         else
+         {
+             System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "NP)");
+         }
+     }
+
+
+     /**
+      *  Gets and returns connections from and to a connection pool for a while.
+      **/
+     public void runTest()
+     {
+         boolean ok = true;
+         try
+         {
+             System.out.println("Started test run at " + new Date());
+
+             if (DEBUG) System.out.println("Checking health just after datasource creation (we expect
+ that the pool does not exist yet) ...");
+             if (mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool exists prior to first getConnection().");
+             }
+
+             // Verify some setters/getters for JDBC properties.
+             System.out.println("Verifying setters/getters ...");
+
+             mds_.setAccess("read only");
+             if (!mds_.getAccess().equals("read only")) {
+                 ok = false;
+                 System.out.println("\nERROR: getAccess() returned unexpected value:
+ |"+mds_.getAccess()+"|");
+             }
+
+             boolean oldBool = mds_.isBigDecimal();
+             boolean newBool = (oldBool ? false : true);
+             mds_.setBigDecimal(newBool);
+             if (mds_.isBigDecimal() != newBool) {
+                 ok = false;
+                 System.out.println("\nERROR: isBigDecimal() returned unexpected value:
+ |"+mds_.isBigDecimal()+"|");
+             }
+             mds_.setBigDecimal(oldBool);
+
+             int oldInt = mds_.getBlockCriteria();
```

```
+             int newInt = (oldInt == 2 ? 1 : 2);
+             mds_.setBlockCriteria(newInt);
+             if (mds_.getBlockCriteria() != newInt) {
+                 ok = false;
+                 System.out.println("\nERROR: getBlockCriteria() returned unexpected value:
+ |"+mds_.getBlockCriteria()+"|");
+             }
+             mds_.setBlockCriteria(oldInt);
+
+             // Verify some setters and getters for socket properties.
+
+             oldBool = mds_.isKeepAlive();
+             newBool = (oldBool ? false : true);
+             mds_.setKeepAlive(newBool);
+             if (mds_.isKeepAlive() != newBool) {
+                 ok = false;
+                 System.out.println("\nERROR: isKeepAlive() returned unexpected value:
+ |"+mds_.isKeepAlive()+"|");
+             }
+             mds_.setKeepAlive(oldBool);
+
+             oldInt = mds_.getReceiveBufferSize();
+             newInt = (oldInt == 256 ? 512 : 256);
+             mds_.setReceiveBufferSize(newInt);
+             if (mds_.getReceiveBufferSize() != newInt) {
+                 ok = false;
+                 System.out.println("\nERROR: getReceiveBufferSize() returned unexpected value:
+ |"+mds_.getReceiveBufferSize()+"|");
+             }
+             mds_.setReceiveBufferSize(oldInt);
+
+
+             System.out.println("CONNECTION 1");
+             Object o = dataSource_.getConnection();
+            System.out.println(o.getClass());
+            System.out.println("******LOOK ABOVE*******");
+             Connection c1 = dataSource_.getConnection();
+
+             if (DEBUG) displayConnectionType(c1, true);
+
+             if (DEBUG) System.out.println("Checking health after first getConnection() ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after first getConnection().");
+             }
+
+             if (!TESTING_THREAD_SAFETY)
+             {
+                 try
+                 {
+                     c1.setAutoCommit(false);
+                     if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
+                     Statement s = c1.createStatement();
+                     ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
+                     while (rs.next ()) {
+                         if (DEBUG) System.out.println(rs.getString(2));
+                     }
+                     rs.close();
+                                     s.close();
+                 }
+                 catch (Exception e) {
+                     e.printStackTrace();
+                     if (DEBUG) System.out.println("Checking health after fatal connection error
+ ...");
+                     if (!mds_.checkPoolHealth(true)) {
+                         ok = false;
+                         System.out.println("\nERROR: Pool is not healthy after fatal connection
+ error.");
+                     }
+                 }
+             }
+
+
+             System.out.println("CONNECTION 2");
+             Connection c2 = dataSource_.getConnection(userid, password);
+             if (DEBUG) displayConnectionType(c2, false);
+             System.out.println("CONNECTION 3");
+             Connection c3 = dataSource_.getConnection();
+             if (DEBUG) displayConnectionType(c3, true);
```

```
+             c1.close();
+
+             if (DEBUG) System.out.println("Checking health after first close() ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after first close().");
+             }
+
+             System.out.println("CONNECTION 4");
+             Connection c4 = dataSource_.getConnection();
+             if (DEBUG) displayConnectionType(c4, true);
+
+             c1.close();  // close this one again
+             c2.close();
+             c3.close();
+             c4.close();
+
+             if (DEBUG) System.out.println("Checking health after last close() ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after last close().");
+             }
+
+             // Start the test daemons.
+             System.out.println("Starting test daemons");
+             startThreads();
+
+             // Run the test daemons for a while; check pool health periodically.
+
+             long startTime = System.currentTimeMillis();
+             long endTime = startTime + timeToRunDaemons_;
+             while (System.currentTimeMillis() < endTime)
+             {
+                 System.out.print("h");
+                 // Let the daemons run for a while, then check pool health.
+                 try {
+                     Thread.sleep(poolHealthCheckCycle_);
+                 }
+                 catch (InterruptedException ie) {}
+                 if (!mds_.checkPoolHealth(true)) {
+                     ok = false;
+                     System.out.println("\nERROR: Pool is not healthy after test daemons started.");
+                 }
+             }
+
+             // Stop the test daemons.
+             System.out.println("\nStopping test daemons");
+             stopThreads();
+
+             if (DEBUG) System.out.println("Checking health after connectionGetter daemons have run
+ ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after test daemons stopped.");
+             }
+
+
+             if (!TESTING_THREAD_SAFETY)
+             {
+                 System.out.println("CONNECTION 5");
+                 Connection c = dataSource_.getConnection();
+                 if (DEBUG) displayConnectionType(c, true);
+                 c.setAutoCommit(false);
+                 if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
+                 Statement s = c.createStatement();
+                 ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
+                 while (rs.next ()) {
+                     if (DEBUG) System.out.println(rs.getString(2));
+                 }
+                 rs.close();
+                                 s.close();
+                 c.close();            }
+
+
+
+             System.out.println("\nClosing the pool...");
+             mds_.closePool();
+
+             if (DEBUG) System.out.println("Checking health after pool closed ...");
```

```
+             Trace.setTraceJDBCOn(true);  // make sure the final stats get printed out
+             Trace.setTraceOn(true);
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after pool closed.");
+             }
+
+             System.out.println();
+             if(ok==true)
+                 System.out.println("test ran ok");
+             else
+                 System.out.println("test failed");
+
+         }
+         catch (Exception e)
+         {
+             System.out.println(e);
+             e.printStackTrace();
+         }
+         finally
+         {
+             System.out.println("Ended test at " + new Date());
+         }
+     }
+
+
+
+     void startThreads()
+     {
+
+         // Create a bunch of threads that call getConnection().
+         Thread[] threads = new Thread[numDaemons_];
+         for (int i=0; i<numDaemons_; i++)
+         {
+             ConnectionGetter getter;
+             // Flip a coin to see if this daemon will specify the default uid, or unique uid.
+             if (random_.nextBoolean())
+             {
+                 getter = new ConnectionGetter(userid,password);
+                 if (TESTING_THREAD_SAFETY) {  // we can use fictional userid
+                     getter = new ConnectionGetter("Thread"+i, "Pwd"+i);
+                 }
+                 else {  // must use a real userid
+                     getter = new ConnectionGetter(userid,password);
+                 }
+             }
+             else
+             {
+                 getter = new ConnectionGetter(null, null);
+             }
+             threads[i] = new Thread(getter, "["+i+"]");
+             threads[i].setDaemon(true);
+         }
+
+         // Start the threads.
+         for (int i=0; i<numDaemons_; i++)
+         {
+             threads[i].start();
+         }
+     }
+
+     void stopThreads()
+     {
+         // Tell the threads to stop.
+         keepDaemonsAlive_ = false;
+         synchronized (daemonSleepLock_) {
+             daemonSleepLock_.notifyAll();
+         }
+
+         // Wait for the daemons to stop.
+         try {
+             Thread.sleep(3000);
+         }
+         catch (InterruptedException ie) {}
+     }
+
+
+
+     // ConnectionGetter ---------------------------------------------------------------
```

```
+
+     /**
+          Helper class. This daemon wakes up at random intervals and either
+          gets another connection from the connection pool or returns a previously-gotten connection
+ to the pool.
+      **/
+     private final class ConnectionGetter implements Runnable
+     {
+         private String uid_;
+         private String pwd_;
+         private boolean useDefaultUid_;
+         private long maxSleepTime_;
+         private String threadName_;
+         private boolean firstConnection_ = true;
+         ArrayList connections_ = new ArrayList();
+         // list of connections that this getter currently 'owns'.
+
+         ConnectionGetter(String uid, String pwd) {
+             uid_ = uid;
+             pwd_ = pwd;
+             if (uid_ == null) useDefaultUid_ = true;
+             else useDefaultUid_ = false;
+             maxSleepTime_ = daemonMaxSleepTime_;  // our own copy that we can adjust
+         }
+
+         public void run( )
+         {
+             threadName_ = Thread.currentThread().getName();
+             if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Starting up");
+
+             try
+             {
+                 while (keepDaemonsAlive_)
+                 {
+                     try
+                     {
+                         // Pick a random sleep-time, between min and max values.
+                         long sleepTime = Math.max((long)(maxSleepTime_ * random_.nextFloat()),
+                                 daemonMinSleepTime_);
+                         // Note: Must never call wait(0), because that waits forever.
+                         synchronized (daemonSleepLock_) {
+                             try {
+                                 daemonSleepLock_.wait(sleepTime);
+                                 System.out.print(".");
+                             }
+                             catch (InterruptedException ie) {}
+                         }
+                         if (!keepDaemonsAlive_) break;
+
+                         // Decide by chance whether to request another connection or return a
+                         // previously-obtained connection.
+                         Connection conn;
+                         if (random_.nextBoolean())  // Leave the decision to chance.
+                         { // Request another connection.
+                             if (useDefaultUid_)
+                             {
+                                 if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") -
+ get()");
+                                 conn = dataSource_.getConnection();
+                             }
+                             else
+                             {
+                                 if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") -
+ get("+uid_+",***)");
+                                 conn = dataSource_.getConnection(uid_, pwd_);
+                             }
+
+                             if (conn == null) {
+                                 System.out.println("ConnectionGetter("+threadName_+") ERROR:
+ getConnection() returned null");
+                             }
+                             else
+                             {
+                                 // Occasionally "forget" that we own a connection, and neglect to
+                                 // close it.
+                                 // Orphaned connections should eventually exceed their maximum
+                                 // lifetime and get "reaped" by the connection manager.
+                                 float val = random_.nextFloat();
+                                 if (firstConnection_ || val < 0.1) { // 'orphan' a few gotten
```

```
+ соединения                                    firstConnection_ = false;
+                                 }
+                                 else {
+                                     connections_.add(conn);
+                                 }
+                                 if (DEBUG) displayConnectionType(conn, useDefaultUid_);
+                                 if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
+                                 {  // We got a pooled connection.  Try speeding up our cycle time.
+                                     if (maxSleepTime_ &gt; 100) maxSleepTime_--;
+                                     else maxSleepTime_ = 100;
+                                 }
+                                 else
+                                 {  // We didn't get a pooled connection.  That means that the pool
+                                    // must be at capacity.  Slow down our cycle time a bit.
+                                     maxSleepTime_ = maxSleepTime_ + 50;
+                                 }
+                             }
+                         }
+                         else { // Close a connection that we currently own.
+                             if (connections_.size() != 0) {
+                                 conn = (Connection)connections_.remove(0);
+                                 conn.close();
+                             }
+                         }
+
+                     }  // inner try
+                     catch (Exception e)
+                     {
+                         e.printStackTrace();
+                     }
+                 } // outer while
+             }  // outer try
+             finally
+             {
+                 if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Stopping");
+                 // Return all the connections that I still have in my list.
+                 for (int i=0; i<connections_.size(); i++) {
+                     Connection conn = (Connection)connections_.remove(0);
+                     try { conn.close(); } catch (Exception e) { e.printStackTrace(); }
+                 }
+
+             }
+         }
+
+     }  // internal class 'ConnectionGetter'
+
+
+ }
```

+ Пример: Применение классов FieldDescription, RecordFormat и Record

The following examples show how you can use the IBM Toolbox for Java FieldDescription, RecordFormat and Record classes with data queues.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Пример: Применение классов FieldDescription

Классы FieldDescription служат для описания различных типов данных, составляющих запись в очереди данных. В приведенных ниже примерах предполагается, записи очереди данных имеют следующий формат:

```
Номер сообщения Отправитель Время отправки  Текст сообщения  Необходимость ответа
      |             |            |               |                 |
   bin(4)        char(50)    char(8)         char(1024)        char(1)
```

```
// Создание описания полей для входных данных
BinaryFieldDescription msgNumber = new BinaryFieldDescription(new AS400Bin4(),
                                   "msgnum");
CharacterFieldDescription sender = new CharacterFieldDescription(new AS400Text(50),
                                   "sender");
CharacterFieldDescription timeSent = new CharacterFieldDescription(new AS400Text(8),
```

```
                                      "timesent");
CharacterFieldDescription msgText = new CharacterFieldDescription(new AS400Text(1024),
                                     "msgtext");
CharacterFieldDescription replyRequired = new CharacterFieldDescription(new AS400Text(1),
                                          "replyreq");
```

Применение класса RecordFormat

Класс RecordFormat предназначен для описания данных, составляющих запись очереди данных.

Пример: Определение и динамическое использование формата записи RecordFormat

В следующем примере с помощью класса RecordFormat создается описание формата записи очереди данных, которое затем применяется для получения записи.

```
RecordFormat entryFormat = new RecordFormat();
// Описание полей записи очереди данных
entryFormat.addFieldDescription(msgNumber);
entryFormat.addFieldDescription(sender);
entryFormat.addFieldDescription(timeSent);
entryFormat.addFieldDescription(msgText);
entryFormat.addFieldDescription(replyRequired);

// Получение записи с помощью созданного формата
Record rec = entryFormat.getNewRecord();
```

Пример: Статическое определение формата RecordFormat

В следующем примере формат записи определяется статически, что позволяет использовать его в нескольких программах.

```
public class MessageEntryFormat extends RecordFormat
{
  // Описания полей
  static BinaryFieldDescription msgNumber = new BinaryFieldDescription(new AS400Bin4(),
                                            "msgnum");
  static CharacterFieldDescription sender = new CharacterFieldDescription(new AS400Text(50),
                                            "sender");
  static CharacterFieldDescription timeSent = new CharacterFieldDescription(new AS400Text(8),
                                               "timesent");
  static CharacterFieldDescription msgText = new CharacterFieldDescription(new AS400Text(1024),
                                             "msgtext");
  static CharacterFieldDescription replyRequired = new CharacterFieldDescription(new AS400Text(1),
                                                   "replyreq");

  public MessageEntryFormat()
  {
    // Выбор имени для формата
    super("MessageEntryFormat");
    // Добавление описаний полей
    addFieldDescription(msgNumber);
    addFieldDescription(sender);
    addFieldDescription(timeSent);
    addFieldDescription(msgText);
    addFieldDescription(replyRequired);
  }
}
```

Пример: Применение статически определенного формата RecordFormat

В следующем примере показано применение статически определенного формата записи RecordFormat в программе на Java:

```
  MessageEntryFormat entryFormat = new MessageEntryFormat();
  // Получение записи с помощью созданного формата
  Record rec = entryFormat.getNewRecord();
```

Применение класса Record

Класс Record обеспечивает доступ к отдельным полям записей очереди данных.

Пример: Применение шаблона объекта Record

```
  // Создание экземпляра объекта очереди данных
  DataQueue dq = new DataQueue(new AS400(), "/qsys.lib/mylib.lib/myq.dtaq");

  // Чтение записи
  DataQueueEntry dqEntry = null;
  try
  {
    dqEntry = dq.read();
  }
  catch(Exception e)
  {
    // Обработка исключительных ситуаций
  }

  // Получение объекта записи из формата записи и
  // инициализация его данными прочитанной записи.
  Record rec = entryFormat.getNewRecord(dqEntry.getData());

  // Вывод всей записи в виде объекта String. Содержимое записи преобразуется
// в объекты Java с помощью формата записи.
  System.out.println(rec.toString());
  // Получение содержимого отдельных полей записи.
// Преобразование содержимого полей в объекты Java.
  Integer num = (Integer)rec.getField(0);   // Получение содержимого по индексу
  String s = (String)rec.getField("sender");// Получение содержимого по имени поля
  String text = (String)rec.getField(3);    // Получение текста сообщения
  // Вывод данных
  System.out.println(num + " " + s + " " + text);
```

Пример: Применение статического объекта Record

Статически определенный объект Record может использоваться с определенным форматом очереди данных, что позволяет применять для полей методы get() и set() с названиями, более информативными, чем getField() и setField(). Статически определенный специальный объект Record позволяет получать вместо объектов пользовательские и базовые типы Java.

Обратите внимание, что в этом примере необходимо явно преобразовать данные в объект Java.

```
  public class MessageEntryRecord extends Record
  {
    static private RecordFormat format = new MessageEntryFormat();

    public MessageEntryRecord()
    {
      super(format);
    }

    public int getMessageNumber()
    {
      // Возврат номера сообщения с типом int. Примечание: Известен как формат
// записи, так и имена полей. Рекомендуется обращаться к полям по их именам
// на случай добавления дополнительных полей.
      return ((Integer)getField("msgnum")).intValue();
    }
```

```
    public String getMessageText()
    {
      // Возврат текста сообщения
      return (String)getField("msgtext");
    }

    public String getSender()
    {
      // Возврат отправителя сообщения
      return (String)getField("sender");
    }

    public String getTimeSent()
    {
      // Возврат отправителя сообщения
      return (String)getField("timesent");
    }

    // Здесь можно добавить команды присвоения значений
  }
```

Пример: Применение класса MessageEntryRecord

Для возврата нового объекта MessageEntryRecord необходимо переопределить метод getNewRecord() в классе MessageEntryFormat (в приведенном выше примере). Для этого необходимо добавить в класс MessageEntryFormat следующий код:

```
    public Record getNewRecord(byte[] data)
    {
      Record r = new MessageEntryRecord();
      r.setContents(data);
      return r;
    }
```

После добавления метода getNewRecord() объект MessageEntryRecord позволит интерпретировать запись очереди данных:

```
  // Получение объекта записи из формата записи и
  // инициализация его данными прочитанной записи.
// Обратите внимание на то, что применяется переопределенный метод getNewRecord().
  MessageEntryRecord rec = (MessageEntryRecord)entryFormat.getNewRecord(dqEntry.getData());

  // Вывод всей записи в виде объекта String. Содержимое записи преобразуется
// в объекты Java с помощью формата записи.
  System.out.println(rec.toString());
  // Получение содержимого отдельных полей записи.
// Преобразование содержимого полей в объекты Java.
  int num = rec.getMessageNumber();   // Получение номера сообщения с типом int
  String s = rec.getSender();         // Получение отправителя
  String text = rec.getMessageText(); // Получение текста сообщения
  // Вывод данных
  System.out.println(num + " " + s + " " + text);
```

Пример: Применение классов DataQueue для занесения записей в очередь данных

This example uses the Record and Record format classes to put data on the queue. String data is converted from Unicode to ebcdic and numbers are converted from Java to the system format. Because data is converted the data queue, entries can be read by a server program, an System i5 program, or another Java program.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с очередь данных. Данная программа применяет класс DataQueue
```

```
// для записи данных в очередь.
//
// В этом примере для занесения данных в очередь применяются классы
// Record и RecordFormat. Строки преобразуются из Unicode в EBCDIC,
// and numbers are converted from Java to the server format.  Because data
// is converted the data queue, entries can be read by a server program,
// an IBM System i Access for Windows program,
// or another Java program.
//
// Это - часть программы производитель-потребитель, отвечающая за работу производителя. Она
// помещает элементы в очередь для потребителя.
//
// Формат вызова:
//    DQProducerExample система
//
////////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQProducerExample extends Object
{
   // Создание программы чтения, принимающей ввод пользователя.
   static BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);
public static void main(String[] parameters)
   {
      System.out.println( " " );

      // Если имя системы не указано - вывод справки и завершение работы.
      if (parameters.length >= 1)
      {
         try
         {
             // Первый параметр - имя системы, содержащей очередь данных.
             String system           = parameters[0];

             // Создание объекта AS400 для сервера, на котором находится очередь данных.
             AS400 as400 = new AS400(system);

             // Создание формата записи очереди данных.
             // This format matches the format in the DQConsumer class.  A              // Запись состоит из:
             //    - четырехбайтового числа -- номера клиента
             //    - четырехбайтового числа -- номера компонента
             //    - 20-символьной строки -- описания компонента
             //    - четырехбайтового числа -- количества компонентов в заказе
             // Создание основных типов данных.
             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");

             // Создание формата записи и добавление в него основных типов.
             RecordFormat dataFormat = new RecordFormat();
             dataFormat.addFieldDescription(customerNumber);
             dataFormat.addFieldDescription(partNumber);
             dataFormat.addFieldDescription(partName);
             dataFormat.addFieldDescription(quantity);
```

```
            // Создание библиотеки, содержащей очередь данных, с помощью класса
            // CommandCall.
            CommandCall crtlib = new CommandCall(as400);
            crtlib.run("CRTLIB JAVADEMO");

            // Создание объекта очереди данных.
            DataQueue dq = new DataQueue(as400, "/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");

            // Создание очереди данных на случай, если программа запущена впервые.
            // Если очередь уже существует, полученное исключение
            // игнорируется.
            try
            {
               dq.create(96);
            }
            catch (Exception e) {};

            // Получение первого поля данных от пользователя.
            System.out.print("Введите номер пользователя (0 для выхода из программы): ");
            int customer = getInt();

            // До тех пор, пока пользователь вводит данные.
            while (customer > 0)
            {
               // Получение остальных данных о заказе от пользователя.
               System.out.print("Введите номер компонента: ");
               int part = getInt();

               System.out.print("Введите количество: ");
               int quantityToOrder = getInt();

               String description = "part " + part;

               // Создание записи с заданным форматом. В данный момент
// запись пуста, она будет заполнена позднее.
               Record data = new Record(dataFormat);

               // Помещение значений, полученных от пользователя, в запись.
               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);

               // Преобразование записи в массив байтов. В очередь
               // данных помещается именно массив байтов.
               byte [] byteData = data.getContents();

               System.out.println("");
               System.out.println("Сохранение записи на сервере...");
               System.out.println("");

               // Добавление записи в очередь данных.
               dq.write(byteData);

               // Получение следующего значения от пользователя.
               System.out.print("Введите номер пользователя (0 для выхода из программы): ");
               customer = getInt();
            }
         }
         catch (Exception e)
         {
            // Если в какой-либо операции произошел сбой -
            // вывод сообщения об ошибке.

            System.out.println("Операция над очередью данных не выполнена");
            System.out.println(e);
```

```
         }
      }

      // Если заданы неверные параметры, вывести текст справки.
      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
         System.out.println("  DQKeyedProducer система");
         System.out.println("");
         System.out.println("где");
         System.out.println("");
         System.out.println(" система = Сервер, на котором расположена очередь данных");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println(" DQProducerExample mySystem");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }

   // Функция, принимающая от пользователя строку символов
   // и преобразующая ее в целое число.
   static int getInt()
   {
      int i = 0;
      boolean Continue = true;

      while (Continue)
      {
         try
         {
            String s = inputStream.readLine();

            i = (new Integer(s)).intValue();
            Continue = false;
         }
         catch (Exception e)
         {
            System.out.println(e);
            System.out.print("Please enter a number ==>");
         }
      }

      return i;
   }
}
```

Пример: Применение классов DataQueue для считывания записей из очереди данных

This program uses the Data Queue classes to read entries off a data queue on the server. The entries were put on the queue with the DQProducer example program.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с объектом DataQueue. В данной программе класс DataQueue
// считывает записи из очереди данных сервера. Записи были помещены в очередь
```

```
// в очередь данных сервера программой-примером DQProducer.
//
// Эта программа - часть примера производитель-потребитель, отвечающая
// за работу потребителя. Она считывает записи из очереди для обработки.
//
// Формат вызова:
//    DQConsumerExample система
//
////////////////////////////////////////////////////////////////////////////


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( " " );

      // Если имя системы не задано - вывод справки и завершение работы.
      if (parameters.length >= 1)
      {
         try
         {

             // Первый параметр - имя системы, содержащей очередь данных.
             String system           = parameters[0];

             // Создание объекта AS400 для сервера, на котором находится очередь данных.
             AS400 as400 = new AS400(system);


             // Создание формата записи очереди данных.
             // This format matches the format in the DQProducer class.  A              // Запись состоит из:
             //    - четырехбайтового числа -- номера клиента
             //    - четырехбайтового числа -- номера компонента
             //    - 20-символьной строки -- описания компонента
             //    - четырехбайтового числа -- количества компонентов в заказе

             // Создание основных типов данных.
             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"

             // Создание формата записи и добавление в него основных типов.
             RecordFormat dataFormat = new RecordFormat();

             dataFormat.addFieldDescription(customerNumber);
             dataFormat.addFieldDescription(partNumber);
             dataFormat.addFieldDescription(partName);
             dataFormat.addFieldDescription(quantity);

             // Создание объекта, представляющего очередь данных
             // на сервере.
             DataQueue dq = new DataQueue(as400, "/QSYS.LIB/JAVADEMO.LIB/PRODCONS.DTAQ");
```

```
            boolean Continue = true;

            // Чтение первой записи из очереди. Тайм-аут равен -1,
// то есть программа будет ждать поступления записи бесконечно.
            System.out.println("*** Ожидание обработки записи ***");

            DataQueueEntry DQData = dq.read(-1);

            while (Continue)
            {

               // Запись считана из очереди. Данные помещаются в запись,
// чтобы программа могла получить доступ к полям данных.
               // Кроме того, при этом данные будут преобразованы
// из формата сервера в формат Java.
               Record data = dataFormat.getNewRecord(DQData.getData());

               // Вывод двух значений из записи.
               Integer amountOrdered = (Integer) data.getField("QUANTITY");
               String  partOrdered   = (String)  data.getField("PART_NAME");

               System.out.println("Необходимо " + amountOrdered + " компонентов "
                                  + partOrdered);
               System.out.println(" ");
               System.out.println("*** Ожидание обработки записи ***");

               // Ожидание следующей записи.
               DQData = dq.read(-1);
            }
         }
         catch (Exception e)
         {
             // Если в какой-либо операции произошел сбой -
             // вывод сообщения об ошибке.
             System.out.println("Операция над очередью данных не выполнена");
             System.out.println(e);
         }
      }

      // Если заданы неверные параметры, вывести текст справки.
      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
         System.out.println(" Система DQConsumerExample");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println(" система = Сервер, на котором расположена очередь данных");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println(" DQConsumerExample mySystem");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}
```

Пример работы с типами данных

You can use the IBM Toolbox for Java AS400DataType classes with ProgramCall to supply data for program parameters and to interpret the data returned in program parameters.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Пример: Применение классов AS400DataType с объектом ProgramCall

Приведенный ниже пример содержит информацию об использовании классов AS400DataType для вызова с помощью ProgramCall системного API ″Получить описание элемента″, QUSRMBRD . API QUSRMBRD позволяет получить описания нужных элементов в файле базы данных. Таблицы, указанные после примера, содержат список необходимых параметров QUSRMBRD и типы данных, которые можно получить с помощью этого примера.

```
    // Создание объекта ProgramCall.  Имя программы и список параметров
    // будут определены позже.
    ProgramCall qusrmbrd = new ProgramCall(new AS400());

    // Создание пустого списка параметров программы
    ProgramParameter[] parms = new ProgramParameter[6];

    // Создание класса AS400DataTypes для преобразования исходных параметров типов Java в
    // данные сервера
    AS400Bin4 bin4 = new AS400Bin4();

    // Для каждого параметра, длина которого отличается от остальных,
    // необходим отдельный объект AS400Text,
    // поскольку в классе AS400Text необходимо указывать длину
    // данных.
    AS400Text char8Converter = new AS400Text(8)
    AS400Text char20Converter = new AS400Text(20);
    AS400Text char10Converter = new AS400Text(10);
    AS400Text char1Converter = new AS400Text(1);

    // Задайте значения в списке параметров; с помощью объектов AS400DataType значения
    // Java преобразуются в массивы байтов, содержащие данные сервера.

    // Для выходных параметров достаточно задать только количество возвращаемых
    // байтов
    parms[0] = new ProgramParameter(135);
    parms[1] = new ProgramParameter(bin4.toBytes(new Integer(135)));
    parms[2] = new ProgramParameter(char8Converter.toBytes("MBRD0100"));
    parms[3] = new ProgramParameter(char20Converter.toBytes("MYFILE    MYLIB     "));
    parms[4] = new ProgramParameter(char10COnverter.toBytes("MYMEMBER  "));
    parms[5] = new ProgramParameter(char1Converter.toBytes("0"));

    // Задание имени программы и списка параметров
    qusrmbrd.setProgram("/qsys.lib/qusrmbrd.pgm", parms);

    // Вызов программы
    try
    {
      qusrmbrd.run();
    }
    catch(Exception e)
    {
      // Обработка исключительных ситуаций
    }

    // Получение информации. Заметьте, что это - неформатированные данные сервера.
byte[] receiverVar = parms[0].getOutputData();


  // С помощью этого метода преобразуются дата и время
```

```
    AS400Text char13Converter = new AS400Text(13);

// С помощью этого метода преобразуется текстовое описание
    AS400Text char50Converter = new AS400Text(50);

    // Создание AS400Structure для обработки полученной информации
    AS400DataType[] dataTypeArray = new AS400DataType[11];
    dataTypeArray[0] = bin4;
    dataTypeArray[1] = bin4;
    dataTypeArray[2] = char10Converter;
    dataTypeArray[3] = char10Converter;
    dataTypeArray[4] = char10Converter;
    dataTypeArray[5] = char10Converter;
    dataTypeArray[6] = char10Converter;
    dataTypeArray[7] = char13Converter;
    dataTypeArray[8] = char13Converter;
    dataTypeArray[9] = char50Converter;
    dataTypeArray[10] = char1Converter;
    AS400Structure returnedDataConverter = new AS400Structure(dataTypeArray);


    // Преобразование полученных данных в массив объектов Java с помощью
    // returnedDataConverter
    Object[] qusrmbrdInfo = dataConverter.toObject(receiverVar, 0);


    // Получение размера полученных данных в байтах
    Integer bytesReturned = (Integer)qusrmbrdInfo[0];
    Integer bytesAvailable = (Integer)qusrmbrdInfo[1];
    if (bytesReturned.intValue() != 135)
    {
      System.out.println("Неверный объем возвращенных данных.");
      System.exit(0);
    }
    String fileName = (String)qusrmbrdInfo[2];
    String libName = (String)qusrmbrdInfo[3];
    String mbrName = (String)qusrmbrdInfo[4];
    String fileAttribute = (String)qusrmbrdInfo[5];
    String sourceType = (String)qusrmbrdInfo[6];
    String created = (String)qusrmbrdInfo[7];
    String lastChanged = (String)qusrmbrdInfo[8];
    String textDesc = (String)qusrmbrdInfo[9];
    String isSourceFile = (String)qusrmbrdInfo[10];

    // Вывод полученной информации
    System.out.println(fileName + " " + libName + " " + mbrName + " " +
                       fileAttribute + sourceType + " " + created + " " +
                       lastChanged + " " + textDesc + " " + isSourceFile);
```

В приведенной ниже таблице перечислены обязательные параметры API QUSRMBRD, использованные в предыдущем примере.

Параметр API QUSRMBRD	**Ввод/вывод**	**Тип**	**Описание**
Переменная получателя	Вывод	Char(*)	Символьный буфер, в который будет скопирована возвращаемая информация.
Длина переменной получателя	Ввод	Bin(4)	Длина символьного буфера, представляющего переменную получателя.

Параметр API QUSRMBRD	Ввод/вывод	Тип	Описание
Имя формата	Ввод	Char(8)	Формат, задающий тип получаемой информации. Допустимы следующие значения: • MBRD0100 • MBRD0200 • MBRD0300 В приведенном ниже примере используется значение MBRD0100.
Полное имя файла базы данных	Ввод	Char(20)	Полное имя файла. Это имя файла, дополненное пробелами до 10 символов, и имя библиотеки, дополненное пробелами до 10 символов. Вместо имени библиотеки можно указать значение *CURLIB или *LIBL.
Имя элемента базы данных	Ввод	Char(10)	Имя элемента, дополненное пробелами до 10 символов. Вместо имени можно указать значение *FIRST или *LAST.
Флаг обработки переопределений	Ввод	Char(1)	Указывает, обрабатываются ли переопределения. 0 указывает, что переопределения не обрабатываются. В данном примере будет применяться именно это значение.

В следующей таблице перечислены значения, получаемые программой из примера (в формате MBRD0100, в соответствии с предыдущим примером):

Получаемое значение	Тип
Байт возвращено	Bin(4)
Байт доступно	Bin(4)
Имя файла базы данных	Char(10)
Имя библиотеки файла базы данных	Char(10)
Имя элемента	Char(10)
Атрибут файла (тип файла: PF, LF, DDMF)	Char(10)
Исходный тип (если это исходный файл, то тип исходного элемента)	Char(10)
Дата и время создания	Char(13)
Дата и время последнего изменения исходных данных	Char(13)
Описание элемента	Char(50)
Флаг исходного файла (0=файл данных, 1=исходный файл)	Char(1)

Пример: Применение класса KeyedDataQueue

This program uses the KeyedDataQueue class to put records on a data queue.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с очередью данных. Данная программа применяет класс KeyedDataQueue
// для записи данных в очередь.
//
// Ключ - это число, а данные - это строка Unicode. В этой программе показан
// один из способов преобразования целого числа в массив байт
// и строки Java в массив байт для того, чтобы ее можно было записать в очередь.
//
// Это - часть программы производитель-потребитель, отвечающая за работу производителя. Она
// помещает элементы в очередь для потребителя.
//
// Формат вызова:
//    DQKeyedProducer система
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQKeyedProducer extends Object
{

   // Создание программы чтения, принимающей ввод пользователя.

   static BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);
public static void main(String[] parameters)
   {

      System.out.println( " " );



      // Если имя системы не указано - вывод справки и завершение работы.

      if (parameters.length >= 1)
      {

         // Первый параметр - имя системы, содержащей очередь данных.

         String system           = parameters[0];



         System.out.println("Приоритет - это числовое значение. Диапазоны допустимых значений:");
         System.out.println("  0 -  49 = низкий приоритет");
         System.out.println(" 50 - 100 = средний приоритет");
         System.out.println("100 +     = высокий приоритет");
         System.out.println(" ");

         try
         {

             // Создание объекта AS400 для сервера, на котором находится очередь данных.

             AS400 as400 = new AS400(system);
```

```
            // Создание библиотеки, содержащей очередь данных, с помощью класса
            // CommandCall.

            CommandCall crtlib = new CommandCall(as400);
            crtlib.run("CRTLIB JAVADEMO");



            // Создание объекта очереди данных.

            QSYSObjectPathName name = new QSYSObjectPathName("JAVADEMO", "PRODCON2", "DTAQ");

            KeyedDataQueue dq = new KeyedDataQueue(as400, name.getPath());



            // Создание очереди данных на случай, если программа запущена впервые.
            // Если очередь уже существует, полученное исключение
            // игнорируется.  Длина ключа - четыре байта, длина
            // записи - 96 байт.

            try
            {
               dq.create(4, 96);
            }
            catch (Exception e) {};



            // Получение данных от пользователя.

            System.out.print("Введите сообщение: ");
            String message = inputStream.readLine();

            System.out.print("Введите приоритет: ");
            int priority = getInt();



            // До тех пор, пока пользователь вводит данные.

            while (priority > 0)
            {

               // Необходимо записать в очередь строку Java.
               // В очередь данных можно записывать только массивы байт,
               // поэтому строку требуется преобразовать в массив.

               byte [] byteData = message.getBytes("UnicodeBigUnmarked");



               // Ключ - это число.  В очередь данных можно записывать
// только массивы байт, поэтому число преобразуется в массив.

               byte [] byteKey = new byte[4];
               byteKey[0] = (byte) (priority >>> 24);
               byteKey[1] = (byte) (priority >>> 16);
               byteKey[2] = (byte) (priority >>> 8);
               byteKey[3] = (byte) (priority);


               System.out.println("");
               System.out.println("Сохранение записи на сервере...");
               System.out.println("");
```

```
                // Добавление записи в очередь данных.

                dq.write(byteKey, byteData);



                // Получение следующего значения от пользователя.

                System.out.print("Введите сообщение: ");
                message = inputStream.readLine();

                System.out.print("Введите приоритет: ");
                priority = getInt();
             }
          }
          catch (Exception e)
          {

             // Если в какой-либо операции произошел сбой -
             // вывод сообщения об ошибке.

             System.out.println("Операция над очередью данных не выполнена");
             System.out.println(e);
          }
       }


       // Если заданы неверные параметры, вывести текст справки.

       else
       {
          System.out.println("");
          System.out.println("");
          System.out.println("");
          System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
          System.out.println(" DQKeyedConsumer система");
          System.out.println("");
          System.out.println("Где");
          System.out.println("");
          System.out.println(" система = Сервер, на котором расположена очередь данных");
          System.out.println("");
          System.out.println("Например:");
          System.out.println("");
          System.out.println("  DQKeyedProducer mySystem");
          System.out.println("");
          System.out.println("");
       }

       System.exit(0);
    }



    // Функция, принимающая от пользователя строку символов
    // и преобразующая ее в целое число.

    static int getInt()
    {
       int i = 0;
       boolean Continue = true;

       while (Continue)
       {
```

```
         try
         {
            String s = inputStream.readLine();

            i = (new Integer(s)).intValue();
            Continue = false;
         }
         catch (Exception e)
         {
            System.out.println(e);
            System.out.print("Please enter a number ==>");
         }
      }

      return i;
   }
}
```

Пример: Применение классов KeyedDataQueue для считывания записей из очереди данных

This program uses the KeyedDataQueue classes to read entries off a data queue on the server. The entries were put on the queue with the DQKeyedProducer example program.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////////
//
// Пример ключевой очереди данных. В этой программе классы KeyedDataQueue применяются для
// чтения записей очереди данных сервера. Записи были помещены в очередь
// программой DQKeyedProducer.
//
// Ключ - это число, а данные - это строка Unicode. В этой программе показан
// один из способов преобразования массива байт в тип int, а также способ
// чтения массива байт и преобразования его в строку Java.
//
// Эта программа - часть примера производитель-потребитель, отвечающая
// за работу потребителя. Она считывает записи из очереди для обработки.
//
// Формат вызова:
//    DQKeyedConsumer система
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import com.ibm.as400.access.*;

public class DQKeyedConsumer extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );

      // Если имя системы не задано - вывод справки и завершение работы.
      if (parameters.length >= 1)
      {

         // Первый параметр - имя системы, содержащей очередь данных.
         String system           = parameters[0];

         // Создание массивов байт для хранения приоритетов:
         // 100 +    = высокий приоритет
         // 50 - 100 = средний приоритет
         //  0 -  49 = низкий приоритет
```

```
        byte [] key0 = new byte[4];
        key0[0] = 0;
        key0[1] = 0;
        key0[2] = 0;
        key0[3] = 0;

        byte [] key50  = new byte[4];
        key50[0] = (byte) (50 >>> 24);
        key50[1] = (byte) (50 >>> 16);
        key50[2] = (byte) (50 >>> 8);
        key50[3] = (byte) (50);

        byte [] key100 = new byte[4];
        key100[0] = (byte) (100 >>> 24);
        key100[1] = (byte) (100 >>> 16);
        key100[2] = (byte) (100 >>> 8);
        key100[3] = (byte) (100);

        try
        {
            // Создание объекта AS400 для сервера, на котором находится очередь данных.
            AS400 as400 = new AS400(system);

            // Создание объекта, представляющего очередь данных
            // на сервере.


            QSYSObjectPathName name = new QSYSObjectPathName("JAVADEMO",
                                                             "PRODCON2",
                                                             "DTAQ");
            KeyedDataQueue dq = new KeyedDataQueue(as400, name.getPath());
            KeyedDataQueueEntry DQData = null;

            try
            {
               boolean Continue = true;

               // Выполнение до остановки программы пользователем.
               while (Continue)
               {
                  // Поиск элемента очереди, который имеет высокий приоритет. Если
                  // такой элемент есть, он обрабатывается. При считывании элемент
                  // удаляется из очереди. Значение тайм-аута для операции
                  // равно 0. Если элемента нет, управляющий элемент возвращается
                  // с пустой записью очереди данных.
                  DQData = dq.read(key100, 0, "GE");

                  if (DQData != null)
                  {
                     processEntry(DQData);
                  }

                  // Если не был найден элемент с высоким приоритетом,
// выполняется поиск элемента с низким приоритетом.
                  else
                  {
                     DQData = dq.read(key50, 0, "GE");

                     if (DQData != null)
                     {
                        processEntry(DQData);
                     }

                     // Если не был найден элемент со средним приоритетом,
                     // выполняется поиск элемента с низким приоритетом.
                     else
```

```
                    {
                       DQData = dq.read(key0, 0, "GE");

                       if (DQData != null)
                       {
                          processEntry(DQData);
                       }

                       else
                       {
                         System.out.println("Нет элементов для обработки,
                         повторная проверка через 30 секунд");
                         Thread.sleep(30000);
                       }
                    }
                 }
              }
           }
           catch (Exception e)
           {
              // Если в какой-либо операции произошел сбой -
              // вывод сообщения об ошибке.
              System.out.println("Не удалось считать элемент из очереди данных.");
              System.out.println(e);
           };

        }
        catch (Exception e)
        {
            // Если в какой-либо операции произошел сбой -
            // вывод сообщения об ошибке.

            System.out.println("Операция над очередью данных не выполнена");
            System.out.println(e);
        }
     }

     // Если заданы неверные параметры, вывести текст справки.
     else
     {
        System.out.println("");
        System.out.println("");
        System.out.println("");
        System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
        System.out.println(" DQKeyedConsumer система");
        System.out.println("");
        System.out.println("Где");
        System.out.println("");
        System.out.println(" система = Сервер, на котором расположена очередь данных");
        System.out.println("");
        System.out.println("Например:");
        System.out.println("");
        System.out.println("");
        System.out.println(" DQKeyedConsumer mySystem");
        System.out.println("");
        System.out.println("");
     }

     System.exit(0);
  }

  static void processEntry(KeyedDataQueueEntry DQData)
  {
     try
     {
         // Получение данных из записи очереди данных.
```

```
// Преобразование полученного массива байт в строку.
String message  = new String(DQData.getData(), "UnicodeBig");

          // Получение ключа из записи очереди данных.
// Преобразование полученного массива байт в число.
          byte [] keyData = DQData.getKey();

          int keyValue = ((keyData[0] & 0xFF) << 24) +
                         ((keyData[1] & 0xFF) << 16) +
                         ((keyData[2] & 0xFF) <<  8) +
                          (keyData[3] & 0xFF);

          // Вывод записи.
          System.out.println("Приоритет: " + keyValue + "   сообщение: " + message);

      }
      catch (Exception e)
      {
         // Если в какой-либо из операций произошел сбой - вывести
         // сообщение о сбое операции над очередью данных и исключение.

         System.out.println("Не удалось считать элемент из очереди данных.")
         System.out.println(e);
      }
   }
}
```

Примеры: Применение класса IFSFile

The following examples show how to use the IBM Toolbox for Java IFSFile class.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

- Пример: Создание каталога
- Пример: Отслеживание ошибок с помощью исключений IFSFile
- Пример: Просмотр списка файлов .txt
- “Пример: Применение метода listFiles() класса IFSFile для просмотра содержимого каталога” на стр. 507

Пример: Создание каталога

```
                      // Создание объекта AS400.
Новый
                      // каталог будет создан в этой
                      // System i5.
     AS400 sys = new AS400("mySystem.myCompany.com");

                      // Создание файлового объекта,
                      // соответствующего каталогу
     IFSFile aDirectory = new IFSFile(sys, "/mydir1/mydir2/newdir");

                      // Создание каталога
     if (aDirectory.mkdir())
        System.out.println("Каталог успешно создан");

                      // В противном случае, создать каталог не удалось
     else
     {
                      // Если объект с таким именем существует,
                      // то - проверка, является он каталогом или
                      // файлом, и вывод соответствующего сообщения
        if (aDirectory.exists())
        {
           if (aDirectory.isDirectory())
              System.out.println("Каталог уже существует");
           else
```

```
            System.out.println("Файл с таким именем уже существует");
      }
      else
         System.out.println("Создать каталог не удалось");
   }

                     // Отключение после завершения
                     // работы с файлами
   sys.disconnectService(AS400.FILE);
```

Пример: Применение исключительных ситуаций IFSFile для отслеживания ошибок

При возникновении ошибки класс IFSFile вызывает исключительную ситуацию ExtendedIOException.Информация об исключительной ситуации включает код возврата, указывающий причину сбоя. Класс IFSFile вызывает исключительные ситуации даже тогда, когда класс из пакета java.io этого не делает. Например, метод удаления, принадлежащий классу java.io.File, возвращает результат операции в виде булевского значения. Соответствующий метод класса IFSFile также возвращает булевское значение, но в случае ошибки, кроме того, вызывает исключительную ситуацию ExtendedIOException, которая передает программе на Java подробную информацию о причинах неудачного удаления.

```
                     // Создание объекта AS400.
   AS400 sys = new AS400("mySystem.myCompany.com");

                     // Создание файлового объекта,
                     // соответствующего файлу
   IFSFile aFile = new IFSFile(sys, "/mydir1/mydir2/myfile");

                     // Удаление файла
   try
   {
      aFile.delete();

                     // Удаление выполнено успешно
      System.out.println("Удаление выполнено успешно");
   }

                     // При удалении возникла ошибка.                          // Получение кода возврата из информации
                     // об исключительной ситуации и вывод сообщения
   catch (ExtendedIOException e)
   {
      int rc = e.getReturnCode();

      switch (rc)
      {
         case ExtendedIOException.FILE_IN_USE:
            System.out.println("Удаление невозможно - файл используется"):
            break;

         case ExtendedIOException.PATH_NOT_FOUND:
            System.out.println("Удаление невозможно - путь не найден"):
            break;

                        // Вывод сообщения для остальных
                        // кодов возврата.

         default:
            System.out.println("Удаление невозможно - rc = ");
            System.out.println(rc);
      }
   }
```

Пример: Просмотр файлов с расширением .txt

Программа на Java может дополнительно задавать критерии соответствия при просмотре файлов в каталоге. Применение такого критерия сокращает число файлов, возвращаемых сервером объекту IFSFile, что повышает производительность. В следующем примере из системы считывается список файлов с расширением .txt:

```
                    // Создание объекта AS400
     AS400 system = new AS400("mySystem.myCompany.com");

                    // Создание файлового объекта
     IFSFile directory = new IFSFile(system, "/");

                    // Создание списка всех файлов
                    // с расширением .txt
     String[] names = directory.list("*.txt");

                    // Вывод результатов
     if (names != null)
       for (int i = 0; i < names.length; i++)
         System.out.println(names[i]);
     else
       System.out.println("Файлы с расширением .txt отсутствуют");
```

Пример: Применение метода listFiles() класса IFSFile для просмотра содержимого каталога

This example program uses the IBM Toolbox for Java IFS classes to list the contents of a directory on the server.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример IFSListFiles. В этой программе для просмотра содержимого каталога сервера
// используются классы интегрированной файловой системы.
//
// Формат вызова:
//    IFSListFiles система каталог
//
// Пример:
//    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        = "";


        // Если указаны не все параметры - вывод справки и завершение работы.

        if (parameters.length >= 2)
        {

            // Первый параметр - имя системы,
            // а второй параметр - имя каталога.
```

```
            system = parameters[0];
            directoryName = parameters[1];

            try
            {
                // Создание объекта AS400 для сервера, содержащего файлы.

                AS400 as400 = new AS400(system);



                // Создание объекта IFSFile для каталога.

                IFSFile directory = new IFSFile(as400, directoryName);


                // Создание списка IFSFiles.  Передача методу listFiles
// фильтра каталога и критерия отбора объектов.
                // Этот метод заносит в кэш атрибуты файлов. Например,
// при вызове метода isDirectory() для объекта IFSFile
                // из полученного массива файлов обращаться к серверу
                // не нужно.
                //
                // Однако при использовании метода listFiles атрибуты в кэше
                // не обновляются автоматически при их изменении на сервере.
                // Это значит, что атрибуты, находящиеся в кэше,
// могут не соответствовать атрибутам сервера.

                IFSFile[] directoryFiles = directory.listFiles(new MyDirectoryFilter(),"*");

                // Если каталог не существует или не содержит данных - вывод сообщения

                if (directoryFiles == null)
                {
                    System.out.println("Каталог не существует");
                    return;
                }

                else if (directoryFiles.length == 0)
                {
                    System.out.println("Каталог пуст");
                    return;
                }

                for (int i=0; i< directoryFiles.length; i++)
                    {
                        // Print out information about list.
                        // Печать имени текущего файла

                        System.out.print(directoryFiles[i].getName());


                        // Выравнивание столбцов вывода

                        for (int j = directoryFiles[i].getName().length(); j <18; j++)
                            System.out.print("   ");


                        // Печать даты последнего изменения файла.

                        long changeDate = directoryFiles[i].lastModified();
                        Date d = new Date(changeDate);
                        System.out.print(d);
                        System.out.print("   ");


                        // Печать типа объекта (файл или каталог)
```

```
                        System.out.print("   ");

                        if (directoryFiles[i].isDirectory())
                            System.out.println("");
                        else
                            System.out.println(directoryFiles[i].length());

                    }
            }

            catch (Exception e)
            {
                // Если при выполнении какой-либо операции возник сбой,
                // появляются сообщение об ошибке и текст исключительной ситуации.

                System.out.println("Операция над списком не выполнена");
                System.out.println(e);
            }
        }

        // Если заданы неверные параметры, вывести текст справки.

        else
        {
            System.out.println("");
            System.out.println("");
            System.out.println("");
            System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
            System.out.println("   IFSListFiles as400 каталог");
            System.out.println("");
            System.out.println("Где");
            System.out.println("");
            System.out.println("   as400  = система, содержащая файлы");
            System.out.println("   каталог = каталог для просмотра");
            System.out.println("");
            System.out.println("Например:");
            System.out.println("");
            System.out.println("   IFSListFiles mySystem /dir1/dir2");
            System.out.println("");
            System.out.println("");
        }

        System.exit(0);
    }
}


////////////////////////////////////////////////////////////////////////////
//
// Класс фильтра каталога печатает информацию из объекта типа "файл".
//
// Фильтр может применяться только для отбора файлов на основании
// информации из объектов файлов. В этом случае обработка списка
// файлов, соответствующих критерию отбора, должна выполняться
// в основной функции.
//
////////////////////////////////////////////////////////////////////////////



class MyDirectoryFilter implements IFSFileFilter
{
    public boolean accept(IFSFile file)
    {
        try
```

```
        {
            // Сохранение записи. Возврат значения true, для того чтобы
// объект IFSList добавил файл в список, возвращаемый
            // методу .list().

            return true;
        }

        catch (Exception e)
        {
            return false;
        }
    }
}
```

Пример: Применение классов IFS для копирования файла из одного каталога в другой

This program uses the installable file system classes to copy a file from one directory to another on the server.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта IFSCopyFile. Для копирования файла из одного
// каталога сервера в другой в программе применяются дополнительные классы
// файловой системы.
//
// Формат вызова:
//    IFSCopyFile система исходный-путь-к-файлу целевой-путь-к-файлу
//
// Пример:
//    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;

      // Если указаны не все параметры - вывод справки и завершение работы.

      if (parameters.length > 2)
      {

         // Первый параметр - имя системы,
         // второй - исходный путь к файлу,
         // третий - целевой путь к файлу.

         system = parameters[0];
```

```
sourceName = parameters[1];
targetName = parameters[2];

try
{
   // Создание объекта AS400 для сервера, содержащего файлы.

   AS400 as400 = new AS400(system);



   // Открытие исходного файла в режиме исключительного доступа.

   source = new IFSFileInputStream(as400, sourceName, IFSFileInputStream.SHARE_NONE);

   System.out.println("Исходный файл успешно открыт");



   // Открытие целевого файла в режиме исключительного доступа.

   target = new IFSFileOutputStream(as400, targetName, IFSFileOutputStream.SHARE_NONE, false);

   System.out.println("Целевой файл успешно открыт");



   // Чтение первых 64 килобайт из исходного файла.

   int bytesRead = source.read(buffer);



   // До тех пор пока в исходном файле есть данные -
   // копировать их в целевой файл.

   while (bytesRead > 0)
   {
      target.write(buffer, 0, bytesRead);
      bytesRead = source.read(buffer);
   }

   System.out.println("Данные успешно скопированы");



   // Закрытие исходного и целевого файлов.

   source.close();
   target.close();



   // Получение даты и времени последнего изменения исходного
   // файла и установка этих атрибутов для целевого файла.

   IFSFile src = new IFSFile(as400, sourceName);
   long dateTime = src.lastModified();

   IFSFile tgt = new IFSFile(as400, targetName);
   tgt.setLastModified(dateTime);

   System.out.println("Для целевого файла успешно установлены дата и время");
   System.out.println("Копирование выполнено");

}
catch (Exception e)
```

```
         {
            // Если при выполнении какой-либо операции произошел сбой -
            // появляется сообщение об ошибке и текст исключительной ситуации.

            System.out.println("Копирование не выполнено");
            System.out.println(e);
         }
      }

      // Если заданы неверные параметры, вывести текст справки.

      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
         System.out.println("   IFSCopyFile as400 исходный целевой");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println("   as400  = система, содержащая файлы");
         System.out.println("   source = имя исходного файла в формате /путь/путь/имя");
         System.out.println("   target = имя целевого файла в формате /путь/путь/имя");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println("   IFSCopyFile myAS400  /dir1/dir2/a.txt  /dir3/b.txt");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}
```

Пример: Применение классов IFS для просмотра содержимого каталога

This program uses the integrated file system classes to list the contents of a directory on the server.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример IFSListFile. В этой программе для просмотра содержимого каталога сервера
// применяются классы интегрированной файловой системы.
//
// Формат вызова:
//    IFSList система каталог
//
// Пример:
//    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       = "";


      // Если указаны не все параметры - вывод справки и завершение работы.

      if (parameters.length >= 2)
      {

         // Первый параметр - имя системы,
         // а второй параметр - имя каталога.

         system = parameters[0];
         directoryName = parameters[1];

         try
         {
            // Создание объекта AS400 для сервера, содержащего файлы.

            AS400 as400 = new AS400(system);



            // Создание объекта IFSFile для каталога.

            IFSFile directory = new IFSFile(as400, directoryName);



            // Создание списка имен.  Передача методу list
// фильтра и критерия отбора.
            //
            // В данном примере список обрабатывается в объекте
            // фильтра. Альтернативой является обработка списка
// после его получения от метода list.

            String[] directoryNames = directory.list(new MyDirectoryFilter(),"*");



            // Если каталог не существует или не содержит данных - вывод сообщения

            if (directoryNames == null)
               System.out.println("Каталог не существует");

            else if (directoryNames.length == 0)
               System.out.println("Каталог пуст");
         }

         catch (Exception e)
         {
            // Если при выполнении какой-либо операции возник сбой,
            // появляется сообщение об ошибке и текст исключительной ситуации.

            System.out.println("Операция над списком не выполнена");
            System.out.println(e);
         }
      }

      // Если заданы неверные параметры, вывести текст справки.

      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
```

```
         System.out.println("   IFSList as400 каталог");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println("   as400  = система, содержащая файлы");
         System.out.println("   каталог = каталог для просмотра");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println("   IFSCopyFile mySystem /dir1/dir2");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}


////////////////////////////////////////////////////////////////////////////
//
// Класс фильтра каталога печатает информацию из объекта типа "файл".
//
// Фильтр может применяться только для отбора файлов на основании
// информации из объектов файлов. В этом случае обработка списка
// файлов, соответствующих критерию отбора, должна выполняться
// в основной функции.
//
////////////////////////////////////////////////////////////////////////////



class MyDirectoryFilter implements IFSFileFilter
{
   public boolean accept(IFSFile file)
   {
      try
      {
         // Печать имени текущего файла

         System.out.print(file.getName());



         // Выравнивание столбцов вывода

         for (int i = file.getName().length(); i < 18; i++)
            System.out.print("   ");



         // Печать даты последнего изменения файла.

         long changeDate = file.lastModified();
         Date d = new Date(changeDate);
         System.out.print(d);
         System.out.print("   ");



         // Печать типа объекта (файл или каталог)

         System.out.print("   ");

         if (file.isDirectory())
            System.out.println("<DIR>");
         else
```

```
            System.out.println(file.length());



         // Сохранение записи. Возврат значения true, для того чтобы
// объект IFSList добавил файл в список, возвращаемый
         // методу .list().

            return true;
      }

      catch (Exception e)
      {
         return false;
      }
   }
}
```

Пример: Применение класса JDBCPopulate для создания и заполнения таблицы

This program uses the IBM Toolbox for Java JDBC driver to create and populate a table.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
//////////////////////////////////////////////////////////////////////////////
//
// Пример применения JDBCPopulate.  Эта программа с помощью драйвера JDBC IBM Toolbox for Java
// на создание и заполнение таблицы с помощью драйвера JDBC.
//
// Формат вызова:
//    JDBCPopulate система имя-набора имя-таблицы
//
// Пример:
//    JDBCPopulate MySystem MyLibrary MyTable
//
//////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{



    // Строки для добавления в столбец WORD таблицы.
    private static final String words[]
        = { "One",      "Two",      "Three",    "Four",     "Five",
            "Six",      "Seven",    "Eight",    "Nine",     "Ten",
            "Eleven",   "Twelve",   "Thirteen", "Fourteen", "Fifteen",
            "Sixteen",  "Seventeen","Eighteen", "Nineteen", "Twenty" };



    public static void main (String[] parameters)
    {
        // Проверка входных параметров.
        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Формат:");
            System.out.println("");
            System.out.println("   JDBCPopulate system collectionName tableName");
            System.out.println("");
            System.out.println("");
            System.out.println("Например:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCPopulate MySystem MyLibrary MyTable");
```

```
            System.out.println("");
            return;
        }

        String system           = parameters[0];
        String collectionName   = parameters[1];
        String tableName        = parameters[2];

        Connection connection   = null;

        try {

            // Загрузка драйвера JDBC IBM Toolbox for Java.
            DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

            // Создание соединения с базой данных. Поскольку ИД пользователя
            // и пароль заранее не известны, на экране появится приглашение.
            //
            // Обратите внимание, что в этой программе применяется схема по умолчанию,
            // поэтому не нужно указывать в операторах SQL имя таблицы.
//
            connection = DriverManager.getConnection ("jdbc:as400://" + system + "/" + collectionName);

            // Если таблица уже существует - удаление ее.
            try {
                Statement dropTable = connection.createStatement ();
                dropTable.executeUpdate ("DROP TABLE " + tableName);
            }
            catch (SQLException e)
        {
                // Игнорировать.
            }

            // Создание таблицы.
            Statement createTable = connection.createStatement ();
            createTable.executeUpdate ("CREATE TABLE " + tableName
                + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
                + " SQUAREROOT DOUBLE)");

            // Подготовка таблицы для вставки строк. Так как этот код выполняется
// много раз, лучше использовать метод PreparedStatement и маркеры
            // параметров.
            PreparedStatement insert = connection.prepareStatement ("INSERT INTO "
                + tableName + " (I, WORD, SQUARE, SQUAREROOT) " + " VALUES (?, ?, ?, ?)");

            // Заполнение таблицы.
            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 ();
            }

            // Вывод сообщения о выполнении.
            System.out.println ("Таблица " + collectionName + "." + tableName + " заполнена.");
        }

        catch (Exception e) {
            System.out.println ();
            System.out.println ("ERROR: " + e.getMessage());
        }

        finally
        {

            // Очистка.
```

```
            try {
                if (connection != null)
                    connection.close ();
            }
            catch (SQLException e)
        {
                // Игнорировать.
            }
        }

        System.exit (0);
    }



}
```

Examples: Using AS400JDBCManagedConnectionPoolDataSource class

These examples demonstrate the use of the AS400JDBCManagedConnectionPoolDataSource class. The AS400JDBCManagedConnectionPoolDataSource class simplifies connection pool maintenance by eliminating the need for user applications to implement their own management code.

Примечание: By using the code examples, you agree to the terms of the “Лицензия на исходный код и отказ от обязательств” на стр. 768.

Пример 1

This brief example shows the basic use of the AS400JDBCManagedConnectionPoolDataSource class.

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

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


public class TestJDBCConnPoolSnippet
{

    /**
     * @param args
     */
    void test()
    {
        AS400JDBCManagedConnectionPoolDataSource cpds0 = new AS400JDBCManagedConnectionPoolDataSource();

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


        cpds0.setSavePasswordWhenSerialized(true);

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

        // Set the initial context factory to use.

        System.setProperty(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");


        // Get the JNDI Initial Context.
```

```
+         Context ctx = new InitialContext();
+
+         // Note: The following is an alternative way to set context properties locally:
+         //   Properties env = new Properties();
+         //   env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
+         //   Context ctx = new InitialContext(env);
+
+         ctx.rebind("mydatasource", cpds0);  // We can now do lookups on cpds, by the name "mydatasource".
+
+         // Create a standard DataSource object that references it.
+
+         AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
+         mds0.setDescription("DataSource supporting connection pooling");
+         mds0.setDataSourceName("mydatasource");
+         ctx.rebind("ConnectionPoolingDataSource", mds0);
+
+         DataSource dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
+
+         AS400JDBCManagedDataSource mds_ = (AS400JDBCManagedDataSource)dataSource_;
+
+         boolean isHealthy = mds_.checkPoolHealth(false);  //check pool health
+
+         Connection c = dataSource_.getConnection();
+
+     }
+
+ }
```

+ Пример 2

+ This example shows more details about how to use the AS400JDBCManagedConnectionPoolDataSource class.

```
+ import java.awt.TextArea;
+ import java.io.BufferedReader;
+ import java.io.File;
+ import java.io.FileReader;
+ import java.io.FileInputStream;
+ import java.io.FileOutputStream;
+ import java.io.OutputStream;
+ import java.io.PrintStream;
+ import java.util.Vector;
+ import java.util.Properties;
+
+ import java.sql.Connection;
+ import javax.sql.DataSource;
+ import java.sql.ResultSet;
+ import java.sql.Statement;
+ import javax.naming.*;
+ import java.util.Date;
+ import java.util.ArrayList;
+ import java.util.Random;
+ import com.ibm.as400.access.AS400;
+ import com.ibm.as400.access.AS400JDBCManagedConnectionPoolDataSource;
+ import com.ibm.as400.access.AS400JDBCManagedDataSource;
+ import com.ibm.as400.access.Trace;
+
+
+ public class TestJDBCConnPool
+ {
+
+     private static final boolean DEBUG = false;
+
+     // If you turn this flag on, be sure to also turn on the following flag:
+     //   AS400JDBCConnection.TESTING_THREAD_SAFETY.
+     private static final boolean TESTING_THREAD_SAFETY = false;
+
+     private static String userid;
+     private static String password;
+     private static String host;
+
+     // Note: For consistency, all time values are stored units of milliseconds.
+     private int initialPoolSize_;  // initial # of connections in pool
+     private int minPoolSize_;      // max # of connections in pool
+     private int maxPoolSize_;      // max # of connections in pool
+     private long maxLifetime_;     // max lifetime (msecs) of connections in pool
+     private long maxIdleTime_;     // max idle time (msecs) of available connections in pool
+     private long propertyCycle_;   // pool maintenance frequency (msecs)
+
```

```
+      private int numDaemons_;       // # of requester daemons to create
+      private static long timeToRunDaemons_; // total duration (msecs) to let the daemons run
+      private long daemonMaxSleepTime_;  // max time (msecs) for requester daemons to sleep each cycle
+      private long daemonMinSleepTime_;  // min time (msecs) for requester daemons to sleep each cycle
+      private long poolHealthCheckCycle_;  // # of msecs between calls to checkPoolHealth()
+
+      private boolean keepDaemonsAlive_ = true;  // When this is false, the daemons shut down.
+
+      private DataSource dataSource_;
+      private AS400JDBCManagedDataSource mds_;
+
+      private final Object daemonSleepLock_ = new Object();
+
+      private Random random_ = new Random();
+
+      static
+      {
+          try {
+              Class.forName("com.ibm.as400.access.AS400JDBCDriver");
+          }
+          catch (Exception e) {
+              System.out.println("Unable to register JDBC driver.");
+              System.exit(0);
+          }
+      }
+
+
+
+      public static void main(String[] args)
+      {
+          host = args[0];
+          userid = args[1];
+          password = args[2];
+          timeToRunDaemons_ = (new Integer(args[3])).intValue() * 1000; //milliseconds
+          //args[3]=time to run in seconds
+          TestJDBCConnPool cptest = new TestJDBCConnPool();
+
+          cptest.setup();
+          cptest.runTest();
+      }
+
+
+
+
+      public void setup()
+      {
+          try
+          {
+              if (DEBUG) System.out.println("TESTING_THREAD_SAFETY flag is " + (TESTING_THREAD_SAFETY
+  ? "true" : "false"));
+
+              if (TESTING_THREAD_SAFETY)
+              {
+                  // Adjust values for performing thread-intensive stress testing.
+                  // NOTE: This assumes that the AS400JDBCConnection class has also been modified to
+                                  // not make actual connections to an actual server.
+                  // To do this, edit AS400JDBCConnection.java, changing its TESTING_THREAD_SAFETY
+                                  // flag to 'false', and recompile.
+                  minPoolSize_ = 100;
+                  maxPoolSize_ = 190;
+                  initialPoolSize_ = 150;  // this should get reset to maxPoolSize_
+                  numDaemons_ = 75;
+                  if (timeToRunDaemons_ == 0) {
+                      timeToRunDaemons_ = 180*1000; // 180 seconds == 3 minutes
+                  }
+              }
+              else
+              {  // Set more conservative values, as we'll be making actual connections to an
+                 // actual server, and we don't want to monopolize the server.
+                  minPoolSize_ = 5;
+                  maxPoolSize_ = 15;
+                  initialPoolSize_ = 9;
+                  numDaemons_ = 4;
+                  if (timeToRunDaemons_ == 0) {
+                      timeToRunDaemons_ = 15*1000; // 15 seconds
+                  }
+              }
+              maxLifetime_ = (int)timeToRunDaemons_ / 3;
+              maxIdleTime_ = (int)timeToRunDaemons_ / 4;
```

```
+             propertyCycle_ = timeToRunDaemons_ / 4;
+             poolHealthCheckCycle_ = Math.min(timeToRunDaemons_ / 4, 20*60*1000);
+             // at least once every 20 minutes (more frequently if shorter run-time)
+             daemonMaxSleepTime_ = Math.min(timeToRunDaemons_ / 3, 10*1000);
+             // at most 10 seconds (less if shorter run-time)
+             daemonMinSleepTime_ = 20;  // milliseconds
+
+             if (DEBUG) System.out.println("setup: Constructing
+ AS400JDBCManagedConnectionPoolDataSource (cpds0)");
+             AS400JDBCManagedConnectionPoolDataSource cpds0 = new
+ AS400JDBCManagedConnectionPoolDataSource();
+
+             // Set general datasource properties.  Note that both CPDS and MDS have these
+             // properties, and they might have different values.
+             cpds0.setServerName(host);
+             cpds0.setDatabaseName(host);//iasp can be here
+             cpds0.setUser(userid);
+             cpds0.setPassword(password);
+
+
+             cpds0.setSavePasswordWhenSerialized(true);
+
+             // Set connection pooling-specific properties.
+             cpds0.setInitialPoolSize(initialPoolSize_);
+             cpds0.setMinPoolSize(minPoolSize_);
+             cpds0.setMaxPoolSize(maxPoolSize_);
+             cpds0.setMaxLifetime((int)(maxLifetime_/1000));  // convert to seconds
+             cpds0.setMaxIdleTime((int)(maxIdleTime_/1000));  // convert to seconds
+             cpds0.setPropertyCycle((int)(propertyCycle_/1000));  // convert to seconds
+             //cpds0.setReuseConnections(false);  // don't re-use connections
+
+             // Set the initial context factory to use.
+
+
+             System.setProperty(Context.INITIAL_CONTEXT_FACTORY,
+ "com.sun.jndi.fscontext.RefFSContextFactory");
+
+
+             // Get the JNDI Initial Context.
+             Context ctx = new InitialContext();
+
+             // Note: The following is an alternative way to set context properties locally:
+             //   Properties env = new Properties();
+             //   env.put(Context.INITIAL_CONTEXT_FACTORY,
+ "com.sun.jndi.fscontext.RefFSContextFactory");
+             //   Context ctx = new InitialContext(env);
+
+
+             ctx.rebind("mydatasource", cpds0);
+                          // We can now do lookups on cpds, by the name"mydatasource".
+
+             if (DEBUG) System.out.println("setup: lookup(\"mydatasource\"" + ")");
+          //   AS400JDBCManagedConnectionPoolDataSource cpds1 =
+ (AS400JDBCManagedConnectionPoolDataSource)ctx.lookup("mydatasource");
+            // if (DEBUG) System.out.println("setup: cpds1.getUser() == |" + cpds1.getUser() + "|");
+
+
+             // Create a standard DataSource object that references it.
+
+             if (DEBUG) System.out.println("setup: Constructing AS400JDBCManagedDataSource (mds0)");
+             AS400JDBCManagedDataSource mds0 = new AS400JDBCManagedDataSource();
+             mds0.setDescription("DataSource supporting connection pooling");
+             mds0.setDataSourceName("mydatasource");
+             ctx.rebind("ConnectionPoolingDataSource", mds0);
+
+             if (DEBUG) System.out.println("setup: lookup(\"ConnectionPoolingDataSource\"" + ")");
+             dataSource_ = (DataSource)ctx.lookup("ConnectionPoolingDataSource");
+             //dataSource_.setLogWriter(output_);
+             if (DEBUG) System.out.println("setup: dataSource_.getUser() == |" +
+ ((AS400JDBCManagedDataSource)dataSource_).getUser() + "|");
+
+             mds_ = (AS400JDBCManagedDataSource)dataSource_;
+
+
+         }
+         catch (Exception e)
+         {
+             e.printStackTrace();
+             System.out.println("Setup error during Trace file creation.");
```

```
+         }
+     }
+
+     void displayConnectionType(Connection conn, boolean specifiedDefaultId)
+     {
+         if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
+         {
+             System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "P)");
+         }
+         else
+         {
+             System.out.print("("+ (specifiedDefaultId ? "+" : "-") + "NP)");
+         }
+     }
+
+
+     /**
+      *  Gets and returns connections from and to a connection pool for a while.
+      **/
+     public void runTest()
+     {
+         boolean ok = true;
+         try
+         {
+             System.out.println("Started test run at " + new Date());
+
+             if (DEBUG) System.out.println("Checking health just after datasource creation (we expect
+ that the pool does not exist yet) ...");
+             if (mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool exists prior to first getConnection().");
+             }
+
+             // Verify some setters/getters for JDBC properties.
+             System.out.println("Verifying setters/getters ...");
+
+             mds_.setAccess("read only");
+             if (!mds_.getAccess().equals("read only")) {
+                 ok = false;
+                 System.out.println("\nERROR: getAccess() returned unexpected value:
+ |"+mds_.getAccess()+"|");
+             }
+
+             boolean oldBool = mds_.isBigDecimal();
+             boolean newBool = (oldBool ? false : true);
+             mds_.setBigDecimal(newBool);
+             if (mds_.isBigDecimal() != newBool) {
+                 ok = false;
+                 System.out.println("\nERROR: isBigDecimal() returned unexpected value:
+ |"+mds_.isBigDecimal()+"|");
+             }
+             mds_.setBigDecimal(oldBool);
+
+             int oldInt = mds_.getBlockCriteria();
+             int newInt = (oldInt == 2 ? 1 : 2);
+             mds_.setBlockCriteria(newInt);
+             if (mds_.getBlockCriteria() != newInt) {
+                 ok = false;
+                 System.out.println("\nERROR: getBlockCriteria() returned unexpected value:
+ |"+mds_.getBlockCriteria()+"|");
+             }
+             mds_.setBlockCriteria(oldInt);
+
+             // Verify some setters and getters for socket properties.
+
+             oldBool = mds_.isKeepAlive();
+             newBool = (oldBool ? false : true);
+             mds_.setKeepAlive(newBool);
+             if (mds_.isKeepAlive() != newBool) {
+                 ok = false;
+                 System.out.println("\nERROR: isKeepAlive() returned unexpected value:
+ |"+mds_.isKeepAlive()+"|");
+             }
+             mds_.setKeepAlive(oldBool);
+
+             oldInt = mds_.getReceiveBufferSize();
+             newInt = (oldInt == 256 ? 512 : 256);
+             mds_.setReceiveBufferSize(newInt);
+             if (mds_.getReceiveBufferSize() != newInt) {
```

```
+                ok = false;
+                System.out.println("\nERROR: getReceiveBufferSize() returned unexpected value:
+ |"+mds_.getReceiveBufferSize()+"|");
+            }
+            mds_.setReceiveBufferSize(oldInt);
+
+
+            System.out.println("CONNECTION 1");
+            Object o = dataSource_.getConnection();
+           System.out.println(o.getClass());
+           System.out.println("******LOOK ABOVE*******");
+            Connection c1 = dataSource_.getConnection();
+
+            if (DEBUG) displayConnectionType(c1, true);
+
+            if (DEBUG) System.out.println("Checking health after first getConnection() ...");
+            if (!mds_.checkPoolHealth(true)) {
+                ok = false;
+                System.out.println("\nERROR: Pool is not healthy after first getConnection().");
+            }
+
+            if (!TESTING_THREAD_SAFETY)
+            {
+                try
+                {
+                    c1.setAutoCommit(false);
+                    if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
+                    Statement s = c1.createStatement();
+                    ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
+                    while (rs.next ()) {
+                        if (DEBUG) System.out.println(rs.getString(2));
+                    }
+                    rs.close();
+                                    s.close();
+                }
+                catch (Exception e) {
+                    e.printStackTrace();
+                    if (DEBUG) System.out.println("Checking health after fatal connection error
+ ...");
+                    if (!mds_.checkPoolHealth(true)) {
+                        ok = false;
+                        System.out.println("\nERROR: Pool is not healthy after fatal connection
+ error.");
+                    }
+                }
+            }
+
+
+            System.out.println("CONNECTION 2");
+            Connection c2 = dataSource_.getConnection(userid, password);
+            if (DEBUG) displayConnectionType(c2, false);
+            System.out.println("CONNECTION 3");
+            Connection c3 = dataSource_.getConnection();
+            if (DEBUG) displayConnectionType(c3, true);
+            c1.close();
+
+            if (DEBUG) System.out.println("Checking health after first close() ...");
+            if (!mds_.checkPoolHealth(true)) {
+                ok = false;
+                System.out.println("\nERROR: Pool is not healthy after first close().");
+            }
+
+            System.out.println("CONNECTION 4");
+            Connection c4 = dataSource_.getConnection();
+            if (DEBUG) displayConnectionType(c4, true);
+
+            c1.close();  // close this one again
+            c2.close();
+            c3.close();
+            c4.close();
+
+            if (DEBUG) System.out.println("Checking health after last close() ...");
+            if (!mds_.checkPoolHealth(true)) {
+                ok = false;
+                System.out.println("\nERROR: Pool is not healthy after last close().");
+            }
+
+            // Start the test daemons.
+            System.out.println("Starting test daemons");
```

```
+             startThreads();
+
+             // Run the test daemons for a while; check pool health periodically.
+
+             long startTime = System.currentTimeMillis();
+             long endTime = startTime + timeToRunDaemons_;
+             while (System.currentTimeMillis() < endTime)
+             {
+                 System.out.print("h");
+                 // Let the daemons run for a while, then check pool health.
+                 try {
+                     Thread.sleep(poolHealthCheckCycle_);
+                 }
+                 catch (InterruptedException ie) {}
+                 if (!mds_.checkPoolHealth(true)) {
+                     ok = false;
+                     System.out.println("\nERROR: Pool is not healthy after test daemons started.");
+                 }
+             }
+
+             // Stop the test daemons.
+             System.out.println("\nStopping test daemons");
+             stopThreads();
+
+             if (DEBUG) System.out.println("Checking health after connectionGetter daemons have run
+  ...");
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after test daemons stopped.");
+             }
+
+
+             if (!TESTING_THREAD_SAFETY)
+             {
+                 System.out.println("CONNECTION 5");
+                 Connection c = dataSource_.getConnection();
+                 if (DEBUG) displayConnectionType(c, true);
+                 c.setAutoCommit(false);
+                 if (DEBUG) System.out.println("SELECT * FROM QIWS.QCUSTCDT");
+                 Statement s = c.createStatement();
+                 ResultSet rs = s.executeQuery("SELECT * FROM QIWS.QCUSTCDT");
+                 while (rs.next ()) {
+                     if (DEBUG) System.out.println(rs.getString(2));
+                 }
+                 rs.close();
+                                 s.close();
+                 c.close();            }
+
+
+
+             System.out.println("\nClosing the pool...");
+             mds_.closePool();
+
+             if (DEBUG) System.out.println("Checking health after pool closed ...");
+             Trace.setTraceJDBCOn(true);  // make sure the final stats get printed out
+             Trace.setTraceOn(true);
+             if (!mds_.checkPoolHealth(true)) {
+                 ok = false;
+                 System.out.println("\nERROR: Pool is not healthy after pool closed.");
+             }
+
+             System.out.println();
+             if(ok==true)
+                 System.out.println("test ran ok");
+             else
+                 System.out.println("test failed");
+
+         }
+         catch (Exception e)
+         {
+             System.out.println(e);
+             e.printStackTrace();
+         }
+         finally
+         {
+             System.out.println("Ended test at " + new Date());
+         }
+     }
+
```

```
+
+
+     void startThreads()
+     {
+
+         // Create a bunch of threads that call getConnection().
+         Thread[] threads = new Thread[numDaemons_];
+         for (int i=0; i<numDaemons_; i++)
+         {
+             ConnectionGetter getter;
+             // Flip a coin to see if this daemon will specify the default uid, or unique uid.
+             if (random_.nextBoolean())
+             {
+                 getter = new ConnectionGetter(userid,password);
+                 if (TESTING_THREAD_SAFETY) {  // we can use fictional userid
+                     getter = new ConnectionGetter("Thread"+i, "Pwd"+i);
+                 }
+                 else {  // must use a real userid
+                     getter = new ConnectionGetter(userid,password);
+                 }
+             }
+             else
+             {
+                 getter = new ConnectionGetter(null, null);
+             }
+             threads[i] = new Thread(getter, "["+i+"]");
+             threads[i].setDaemon(true);
+         }
+
+         // Start the threads.
+         for (int i=0; i<numDaemons_; i++)
+         {
+             threads[i].start();
+         }
+     }
+
+     void stopThreads()
+     {
+         // Tell the threads to stop.
+         keepDaemonsAlive_ = false;
+         synchronized (daemonSleepLock_) {
+             daemonSleepLock_.notifyAll();
+         }
+
+         // Wait for the daemons to stop.
+         try {
+             Thread.sleep(3000);
+         }
+         catch (InterruptedException ie) {}
+     }
+
+
+
+     // ConnectionGetter -----------------------------------------------------------------
+
+     /**
+          Helper class. This daemon wakes up at random intervals and either
+          gets another connection from the connection pool or returns a previously-gotten connection
+ to the pool.
+      **/
+     private final class ConnectionGetter implements Runnable
+     {
+         private String uid_;
+         private String pwd_;
+         private boolean useDefaultUid_;
+         private long maxSleepTime_;
+         private String threadName_;
+         private boolean firstConnection_ = true;
+         ArrayList connections_ = new ArrayList();
+         // list of connections that this getter currently 'owns'.
+
+         ConnectionGetter(String uid, String pwd) {
+             uid_ = uid;
+             pwd_ = pwd;
+             if (uid_ == null) useDefaultUid_ = true;
+             else useDefaultUid_ = false;
+             maxSleepTime_ = daemonMaxSleepTime_;  // our own copy that we can adjust
+         }
+
```

```
+         public void run( )
+         {
+             threadName_ = Thread.currentThread().getName();
+             if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Starting up");
+
+             try
+             {
+                 while (keepDaemonsAlive_)
+                 {
+                     try
+                     {
+                         // Pick a random sleep-time, between min and max values.
+                         long sleepTime = Math.max((long)(maxSleepTime_ * random_.nextFloat()),
+                                 daemonMinSleepTime_);
+                         // Note: Must never call wait(0), because that waits forever.
+                         synchronized (daemonSleepLock_) {
+                             try {
+                                 daemonSleepLock_.wait(sleepTime);
+                                 System.out.print(".");
+                             }
+                             catch (InterruptedException ie) {}
+                         }
+                         if (!keepDaemonsAlive_) break;
+
+                         // Decide by chance whether to request another connection or return a
+                         // previously-obtained connection.
+                         Connection conn;
+                         if (random_.nextBoolean())  // Leave the decision to chance.
+                         { // Request another connection.
+                             if (useDefaultUid_)
+                             {
+                                 if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") -
+ get()");
+                                 conn = dataSource_.getConnection();
+                             }
+                             else
+                             {
+                                 if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") -
+ get("+uid_+",***)");
+                                 conn = dataSource_.getConnection(uid_, pwd_);
+                             }
+
+                             if (conn == null) {
+                                 System.out.println("ConnectionGetter("+threadName_+") ERROR:
+ getConnection() returned null");
+                             }
+                             else
+                             {
+                                 // Occasionally "forget" that we own a connection, and neglect to
+                                 // close it.
+                                 // Orphaned connections should eventually exceed their maximum
+                                 // lifetime and get "reaped" by the connection manager.
+                                 float val = random_.nextFloat();
+                                 if (firstConnection_ || val < 0.1) { // 'orphan' a few gotten
+ соединения                                  firstConnection_ = false;
+                                 }
+                                 else {
+                                     connections_.add(conn);
+                                 }
+                                 if (DEBUG) displayConnectionType(conn, useDefaultUid_);
+                                 if (conn instanceof com.ibm.as400.access.AS400JDBCConnectionHandle)
+                                 {  // We got a pooled connection.  Try speeding up our cycle time.
+                                     if (maxSleepTime_ &amp;gt; 100) maxSleepTime_--;
+                                     else maxSleepTime_ = 100;
+                                 }
+                                 else
+                                 {  // We didn't get a pooled connection.  That means that the pool
+                                    // must be at capacity.  Slow down our cycle time a bit.
+                                     maxSleepTime_ = maxSleepTime_ + 50;
+                                 }
+                             }
+                         }
+                         else { // Close a connection that we currently own.
+                             if (connections_.size() != 0) {
+                                 conn = (Connection)connections_.remove(0);
+                                 conn.close();
+                             }
+                         }
+
```

```
                    }  // inner try
                    catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                } // outer while
            }  // outer try
            finally
            {
                if (DEBUG) System.out.println("ConnectionGetter("+threadName_+") Stopping");
                // Return all the connections that I still have in my list.
                for (int i=0; i<connections_.size(); i++) {
                    Connection conn = (Connection)connections_.remove(0);
                    try { conn.close(); } catch (Exception e) { e.printStackTrace(); }
                }

            }
        }

    }  // internal class 'ConnectionGetter'


}
```

Пример: Применение класса JDBCQuery для отправки запроса к таблице

This program uses the IBM Toolbox for Java JDBC driver to query a table and output its contents.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////////
//
// Пример применения JDBCQuery. Эта программа с помощью драйвера JDBC IBM Toolbox for Java
// отправляет запрос в таблицу и выводит ее содержимое.
//
// Формат вызова:
//    JDBCQuery система имя-набора имя-таблицы
//
// Пример:
//    JDBCQuery MySystem qiws qcustcdt
//
////////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{



    // Форматирование строки по заданной ширине.
    private static String format (String s, int width)
    {
        String formattedString;

        // Если строка короче, чем требуется,
        // ее нужно дополнить пробелами
        if (s.length() < width) {
            StringBuffer buffer = new StringBuffer (s);
            for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
            formattedString = buffer.toString();
        }

        // В противном случае строку нужно усечь.
        else
            formattedString = s.substring (0, width);

        return formattedString;
    }
```

```
public static void main (String[] parameters)
{
    // Проверка входных параметров.
    if (parameters.length != 3) {
        System.out.println("");
        System.out.println("Формат:");
        System.out.println("");
        System.out.println("   JDBCQuery system collectionName tableName");
        System.out.println("");
        System.out.println("");
        System.out.println("Например:");
        System.out.println("");
        System.out.println("");
        System.out.println("   JDBCQuery mySystem qiws qcustcdt");
        System.out.println("");
        return;
    }

    String system           = parameters[0];
    String collectionName   = parameters[1];
    String tableName        = parameters[2];

    Connection connection   = null;

    try {

        // Загрузка драйвера JDBC IBM Toolbox for Java.
        DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());

        // Создание соединения с базой данных. Поскольку ИД пользователя
        // и пароль заранее не известны, на экране появится приглашение.
        connection = DriverManager.getConnection ("jdbc:as400://" + system);
        DatabaseMetaData dmd = connection.getMetaData ();

        // Выполнение запроса.
        Statement select = connection.createStatement ();
        ResultSet rs = select.executeQuery (
           "SELECT * FROM " + collectionName + dmd.getCatalogSeparator() + tableName);

        // Получение информации о результатах. Задание
        // ширины столбца равной максимальному из двух значений:
        // длины метки и длины данных.
        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));
        }

        // Вывод заголовков столбцов.
        for (int i = 1; i <= columnCount; ++i) {
            System.out.print (format (rsmd.getColumnLabel(i), columnWidths[i-1]));
            System.out.print("   ");
        }
        System.out.println ();

        // Вывод пунктирной линии.
        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 ();

            // Итерационный блок вывода колонок с результатами
            // для каждого ряда данных.
            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
        {

            // Очистка.
            try {
                if (connection != null)
                    connection.close ();
            }
            catch (SQLException e)
        {
                // Игнорировать.
            }
        }

        System.exit (0);
    }


}
```

Пример: Составление списка заданий с помощью объекта JobList

This program is an example of the Job support in the IBM Toolbox for Java. It lists job identification information for a specific user on the system.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта JobList IBM Toolbox for Java.
// Эта программа показывает информацию о заданиях,
// запущенных указанным пользователем.
//
// Формат вызова:
//    listJobs2 система пользователь пароль
//
///////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;
```

```
public class listJobs2 extends Object
{

                    // Создание объекта для вызова
                    // нестатических методов.
   public static void main(String[] parameters)
   {
      listJobs2 me = new listJobs2();
      me.Main(parameters);

      System.exit(0);
   }



   void Main(String[] parameters)
   {

                    // Если не указана система, то будет выдана справочная информация и работа будет завершена.
      if (parameters.length == 0)
      {
         showHelp();
         return;
      }

                    // Assign the parameters to variables.                      // Первый параметр - имя системы, второй
                    // имя пользователя, третий - пароль.
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
      {

                    // Создание объекта AS400 для указанной системы.
                    // Если пользователь указал имя или пароль -
// передача объекту указанных значений.
         AS400 as400 = new AS400(parameters[0]);

         if (userID != null)
            as400.setUserId(userID);

         if (password != null)
            as400.setPassword(password);



         System.out.println("получение списка ...     ");


                    // Создание объекта JobList. Этот объект позволяет
// получить список активных заданий в системе.
         JobList jobList = new JobList(as400);


                    // Получение списка активных заданий.
```

```
         Enumeration list = jobList.getJobs();


                    // Для каждого задания в списке ...
         while (list.hasMoreElements())
         {

                    // Выборка задания из списка. Если указано имя пользователя,
// и оно совпадает с именем пользователя задания -
                    // вывод информации о задании. Если имя пользователя
// не указано - вывод информации обо всех заданиях.
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("Непредвиденная ошибка");
         e.printStackTrace();
      }
   }



      // Если заданы неверные параметры, вывести текст справки.
   void showHelp()
   {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
         System.out.println("   listJobs2 System UserID Password");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println("   System   = сервер для подключения");
         System.out.println("   UserID   = имя пользователя в этой системе");
         System.out.println("   Password = пароль этого пользователя (необязательный параметр)");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println("   listJobs2 MYAS400 JavaUser pwd1");
         System.out.println("");
         System.out.println("");
   }
}
```

Пример: Получение списка заданий с помощью объекта JobList

This example gets a list of jobs on the server and outputs the job status followed by job identifier using the IBM Toolbox for Java Job classes.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Эта программа демонстрирует работу классов job в
// IBM Toolbox for Java.  Она получает список заданий сервера
// и выводит состояние и идентификатор каждого задания.
//
//
// Формат вызова:
//    listJobs система пользователь пароль
//
// (Идентификатор пользователя и пароль указывать не обязательно)
//
////////////////////////////////////////////////////////////////////////

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)
   {

      // Если не указана система, то будет выдана справочная информация
      // и работа будет завершена.
      if (parameters.length == 0)
      {
         showHelp();
         return;
      }

      // Настроить параметры объекта AS400. Первый параметр (имя системы)
// задается пользователем.  Второй и третий параметры - необязательные.
// Это имя пользователя и пароль. Перед передачей имени и пароля
// в объект AS400 они преобразуются в верхний регистр.
      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
      {

         // Создание по указанному имени системы объекта AS400.
         AS400 as400 = new AS400(parameters[0]);

         // Если указано имя и/или пароль пользователя -
         // передача их объекту AS400.
```

```
         if (userID != null)
            as400.setUserId(userID);

         if (password != null)
            as400.setPassword(password);

         // Создание объекта списка заданий с указанием системы.
JobList jobList = new JobList(as400);

         // Получение списка запущенных в системе заданий.
         Enumeration listOfJobs = jobList.getJobs();

         // Вывод информации обо всех заданиях системы.
         while (listOfJobs.hasMoreElements())
         {
            printJobInfo((Job) listOfJobs.nextElement(), as400);
         }

      }
      catch (Exception e)
      {
         System.out.println("Непредвиденная ошибка");
         System.out.println(e);
      }
   }



   void printJobInfo(Job job, AS400 as400)
   {

      // Создание необходимых объектов преобразования
      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);


      // Имя, номер и другая информация о задании из списка заданий получены.
// Получение дополнительной информации о задании с помощью API сервера.
      try
      {
         // Создание объекта вызова программы
         ProgramCall pgm = new ProgramCall(as400);

         // Вызываемая программа сервера принимает пять параметров
         ProgramParameter[] parmlist = new ProgramParameter[5];

         // Первый параметр - массив байт, содержащий вывод.
         // файл данных Unicode.  Для выходных данных будет выделен буфер в 1 Кб.
         parmlist[0] = new ProgramParameter( 1024 );

         // Второй параметр - размер буфера вывода (1 Кб).
         Integer iStatusLength = new Integer( 1024 );
         byte[]  statusLength = bin4Converter.toBytes( iStatusLength );
         parmlist[1] = new ProgramParameter( statusLength );

         // Третий параметр - имя формата данных.
         // Применяется формат JOBI0200, так как он содержит состояние задания.
         byte[] statusFormat = text8Converter.toBytes("JOBI0200");
         parmlist[2] = new ProgramParameter( statusFormat );

         // Четвертый параметр - это имя задания в формате "имя пользователь номер".
         // Длина имени задания - 10 символов, имени пользователя - 10 символов,
```

```
        // номера - 6 символов. Применение для преобразования и выравнивания
// данных объектов преобразования текста.
        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 );

        // Последний параметр - идентификатор задания. Он будет оставлен пустым.
byte[] jobID = text16Converter.toBytes("                ");
        parmlist[4] = new ProgramParameter( jobID );


        // Запуск программы.
        if (pgm.run( "/QSYS.LIB/QUSRJOBI.PGM", parmlist )==false)
        {
           // Если программа не была выполнена - вывод сообщения об ошибке.
           AS400Message[] msgList = pgm.getMessageList();
           System.out.println(msgList[0].getText());
        }
        else
        {
           // Программа выполнена. Вывод информации о состоянии с ИД каждого
// задания в формате имя-задания.имя-пользователя.идентификатор
           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);
      }

   }



      // Если заданы неверные параметры, вывести текст справки.
   void showHelp()
   {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
         System.out.println("   listJobs System UserID Password");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println("   System   = сервер для подключения");
         System.out.println("   UserID   = имя пользователя в этой системе (необязательный параметр)");
         System.out.println("   Password = пароль этого пользователя (необязательный параметр)");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println("   listJobs MYAS400 JavaUser pwd1");
```

```
        System.out.println("");
        System.out.println("");
   }
}
```

Пример: Просмотр сообщений протокола заданий с помощью объекта JobLog

This program is an example of the job log function of the IBM Toolbox for Java. It will display the messages in the job log for a job that belongs to the current user.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Эта программа демонстрирует работу функции протокола заданий
// IBM Toolbox for Java. Она будет выводить сообщения протокола
// задания, принадлежащего текущему пользователю.
//
// Формат вызова:
//    jobLogExample система пользователь пароль
//
// (Пароль - необязательный параметр)
//
////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.util.*;
import com.ibm.as400.access.*;

public class jobLogExample
{


    public static void main(String[] args)
    {
        // Если система и пользователь не указаны - вывод справки и завершение работы.
        if (args.length < 2)
        {
            System.out.println("Usage:  jobLogExample system userid <password>");
            return;
        }

        String userID     = null;

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке. Если в
            // строке были указаны ИД пользователя и пароль -
            // передача этих значений.
            AS400 system = new AS400 (args[0]);

            if (args.length > 1)
            {
               userID = args[1];
               system.setUserId(userID);
            }

            if (args.length > 2)
               system.setPassword(args[2]);


            // Создание объекта списка заданий. С помощью этого объекта будет
// получен список активных заданий в системе. Получив список,
// программа найдет задание текущего пользователя.
JobList jobList = new JobList(system);
```

```
            // Получение списка активных заданий
            Enumeration list = jobList.getJobs();

            boolean Continue = true;

            // Поиск задания текущего пользователя в списке
            while (list.hasMoreElements() && Continue)
            {
               Job j = (Job) list.nextElement();

               if (j.getUser().trim().equalsIgnoreCase(userID))
               {
                  // Обнаружено задание текущего пользователя. Для него
                  // будет создан объект протокола задания.
                  JobLog jlog = new JobLog(system, j.getName(), j.getUser(), j.getNumber());

                  // Вывод сообщений протокола задания.
                  Enumeration messageList = jlog.getMessages();

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

                  // Выход после вывода сообщений одного из заданий пользователя.
                  Continue = false;
               }
            }
         }
         catch (Exception e)
         {
            System.out.println ("Ошибка: " + e.getMessage ());
         }

         System.exit(0);
      }

}
```

Пример: Создание буферных файлов

This example shows how to create a spooled file on a server from an input stream.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////
//
// В данном примере на сервере создается буферный файл,
// в который записываются данные из потока ввода.
//
///////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;

import com.ibm.as400.access.*;

class NPExampleCreateSplf
{

// Метод для создания буферного файла в указанной системе,
// в указанной очереди вывода из заданного потока ввода.
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();

        // Создание списка параметров PrintParameterList со значениями,
        // переопределяющими параметры принтера по умолчанию.
        // Переопределяется очередь вывода и число копий.
        parms.setParameter(PrintObject.ATTR_COPIES, 4);
        if (outputQueue != null)
        {
           parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outputQueue.getPath());
        }
        out = new SpooledFileOutputStream(system,
                                          parms,
                                          null,
                                          null);

        // Чтение данных из потока ввода до его завершения
        // и передача их в поток вывода буферного файла.
        do
        {
            bytesRead = in.read(buf);
            if (bytesRead != -1)
            {
                out.write(buf, 0, bytesRead);
            }
        } while (bytesRead != -1);

        out.close();  // Закрытие буферного файла

        spooledFile = out.getSpooledFile();   // Получение ссылки на новый буферный файл

    }
    catch (Exception e)
    {
        //...обработка исключительных ситуаций...
    }
    return spooledFile;
}

}
```

Пример: Создание буферных файлов SCS

В этом примере продемонстрировано применение класса SCS3812Writer для создания потока данных SCS и его записи в буферный файл на сервере.

Это приложение поддерживает следующие аргументы, для каждого из которых предусмотрено значение по умолчанию:

- Имя сервера, в котором будет создан буферный файл.
- Имя очереди вывода сервера, в которую будет помещен буферный файл.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример использования класса SCS3812Writer IBM Toolbox for 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;

            // Обработка аргументов.
            if (args.length >= 1)
            {
                system = new AS400(args[0]);    // Создание объекта AS400
            } else {
                system = new AS400(DEFAULT_SYSTEM);
            }

            if (args.length >= 2)               // Set the outq
            {
                parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, args[1]);
            } else {
                parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, DEFAULT_OUTQ);
            }

            out = new SpooledFileOutputStream(system, parms, null, null);

            scsWtr = new SCS3812Writer(out, 37);

            // Запись данных в буферный файл.
            scsWtr.setLeftMargin(1.0);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_5);
            scsWtr.write("          Печать в программе на Java");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setCPI(10);
            scsWtr.write("Этот документ был создан с помощью IBM Toolbox for Java.");
            scsWtr.newLine();
            scsWtr.write("Остальная часть этого документа содержит примеры применения");
            scsWtr.newLine();
            scsWtr.write("класса SCS3812Writer.");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setUnderline(true); scsWtr.write("Установка шрифтов:"); scsWtr.setUnderline(false);
            scsWtr.newLine();
            scsWtr.setFont(scsWtr.FONT_COURIER_10); scsWtr.write(" Шрифт Courier ");
            scsWtr.setFont(scsWtr.FONT_COURIER_BOLD_10); scsWtr.write(" Шрифт Courier полужирный ");
            scsWtr.setFont(scsWtr.FONT_COURIER_ITALIC_10); scsWtr.write(" Шрифт Courier курсив ");
            scsWtr.newLine();
            scsWtr.setBold(true); scsWtr.write("Шрифт Courier полужирный курсив ");
            scsWtr.setBold(false);
            scsWtr.setCPI(10);
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setUnderline(true); scsWtr.write("Строк на дюйм:"); scsWtr.setUnderline(false);
            scsWtr.newLine();
            scsWtr.write("Ниже должны быть напечатаны строки с плотностью 8 строк на дюйм.");
            scsWtr.newLine();
            scsWtr.newLine();
            scsWtr.setLPI(8);
            scsWtr.write("Первая строка"); scsWtr.newLine();
            scsWtr.write("Вторая строка"); scsWtr.newLine();
```

```
            scsWtr.write("Третья строка"); scsWtr.newLine();
            scsWtr.write("Четвертая строка"); scsWtr.newLine();
            scsWtr.write("Пятая строка"); scsWtr.newLine();
            scsWtr.write("Шестая строка"); scsWtr.newLine();
            scsWtr.write("Седьмая строка"); scsWtr.newLine();
            scsWtr.write("Восьмая строка"); scsWtr.newLine();
            scsWtr.endPage();
            scsWtr.setLPI(6);
            scsWtr.setSourceDrawer(1);
            scsWtr.setTextOrientation(0);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.write("Эта страница должна быть напечатана в книжном формате из лотка 1.");
            scsWtr.endPage();
            scsWtr.setSourceDrawer(2);
            scsWtr.setTextOrientation(90);
            scsWtr.absoluteVerticalPosition(6);
            scsWtr.write("Эта страница должна быть напечатана в альбомном формате из лотка 2.");
            scsWtr.endPage();
            scsWtr.close();
            System.out.println("Создан образец буферного файла.");
            System.exit(0);
        }
        catch (Exception e)
        {
            // Обработка ошибок.
            System.out.println("Исключительная ситуация при создании буферного файла. " + e);
            System.exit(0);
        }
    }
}
```

Пример: Чтение буферных файлов

This example shows the use of the PrintObjectInputStream class to read an existing spooled file.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
//////////////////////////////////////////////////////////////////////
//
// Пример получения данных из буферного файла сервера.
//
// Пример применения класса PrintObjectInputStream IBM Toolbox for Java.
//
//////////////////////////////////////////////////////////////////////
        try{
        byte[] buf = new byte[2048];
        int bytesRead;
        AS400 sys = new AS400();
        SpooledFile splf = new SpooledFile( sys,                // AS400
                                            "MICR",       // имя объекта splf
                                            17,                 // номер буферного файла
                                            "QPRTJOB",    // имя задания
                                            "QUSER",      // пользователь задания
                                            "020791" );   // номер задания

        // Открытие буферного файла для чтения и создание потока ввода.
InputStream in = splf.getInputStream(null);

        do
        {
            // Чтение buf.length байт данных из буферного файла в созданный
            // буфер. Возвращаемое значение равно числу считанных байт.
// Данные представляют собой двоичный поток данных принтера,
            // составляющий содержимое буферного файла.
            bytesRead = in.read( buf );
            if( bytesRead != -1 )
            {
```

```
            // Обработка данных буферного файла.
            System.out.println( "Получено " + bytesRead + " байт" );
        }
    } while( bytesRead != -1 );

    in.close();
}
catch( Exception e )
{
    // Исключительная ситуация
}
```

Пример: Чтение и преобразование буферных файлов

Ниже приведены примеры настройки объекта PrintParameterList для различного преобразования данных буферного файла при чтении. В приведенных ниже сегментах кода предполагается, что буферный файл существует на сервере, и метод *createSpooledFile()* создает экземпляр класса SpooledFile, представляющий буферный файл.

Пример объекта PrintObjectPageInputStream

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведен пример создания объекта PrintObjectPageInputStream для чтения страниц данных в виде изображений в формате GIF. В данном случае каждая страница буферного файла будет преобразована в одно изображение. Для преобразования данных применяется объект преобразования в формат GIF рабочей станции.

```
// Создание буферного файла
SpooledFile splF = createSpooledFile();

// Настройка списка параметров печати
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_WORKSTATION_CUST_OBJECT, "/QSYS.LIB/QWPGIF.WSCST");
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*WSCST");

// Создание потока постраничного ввода из буферного файла
PrintObjectPageInputStream is = splF.getPageInputStream(printParms);
```

Пример объекта PrintObjectTransformedInputStream

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведен пример создания объекта PrintObjectTransformedInputStream для чтения данных в формате TIFF. Для преобразования данных применяется объект преобразования в формат TIFF рабочей станции (сжатие G4).

```
// Создание буферного файла
SpooledFile splF = createSpooledFile();

// Настройка списка параметров печати
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_WORKSTATION_CUST_OBJECT, "/QSYS.LIB/QWPTIFFG4.WSCST");
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*WSCST");

// Создание потока преобразованного ввода из буферного файла
PrintObjectTransformedInputStream is = splF.getTransformedInputStream(printParms);
```

Пример объекта PrintObjectTransformedInputStream, использующего данные о производителе и модели

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Ниже приведен пример создания объекта PrintObjectTransformedInputStream для чтения данных, отформатированных для вывода на текстовый принтер. Для преобразования данных указывается изготовитель и модель *HP4.

```
// Создание буферного файла
SpooledFile splF = createSpooledFile();

// Настройка списка параметров печати
PrintParameterList printParms = new PrintParameterList();
printParms.setParameter(PrintObject.ATTR_MFGTYPE, "*HP4");

// Создание потока преобразованного ввода из буферного файла
PrintObjectTransformedInputStream is = splF.getTransformedInputStream(printParms);
```

Пример: Асинхронное создание списка буферных файлов (с помощью обработчиков событий)

This example demonstrates listing all spooled files on a server asynchronously using the PrintObjectListListener interface to get feedback as the list is being built. Listing asynchronously allows the caller to start processing the list objects before the entire list is built for a faster perceived response time for the user.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
//////////////////////////////////////////////////////////////////////
//
// В данном примере список буферных файлов системы создается асинхронно.
// Информация о создании списка будет получена с помощью интерфейса
// PrintObjectListListener. Создание списка в асинхронном режиме
// позволяет начать обработку объектов списка, не дожидаясь окончания
// создания списка. Такой подход уменьшает время ожидания.
//
//////////////////////////////////////////////////////////////////////

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;
    }

    // Просмотр списка буферных файлов системы в асинхронном режиме с помощью обработчика событий
    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(" Получение списка всех буферных файлов в
          асинхронном режиме с помощью обработчика событий");

    SpooledFileList splfList = new SpooledFileList(system_);

    // Настройка фильтров - все пользователи, все очереди
    splfList.setUserFilter("*ALL");
    splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

    // Добавление обработчика событий
    splfList.addPrintObjectListListener(this);

    // Открытие списка с помощью метода openAsynchronously,
    // возвращающего управления немедленно.
    splfList.openAsynchronously();

    do
    {
        // Ожидание составления списка или появления в нем хотя бы 25 объектов
        waitForWakeUp();

        fCompleted = splfList.isCompleted();
        size = splfList.size();

        // Вывод имен всех объектов, добавленных в список
        // с момента последнего вызова программы
        while (listed < size)
        {
            if (fListError)
            {
                System.out.println(" Исключительная ситуация при обработке списка - "
                 + listException);
                break;
            }

            if (fListClosed)
            {
                System.out.println(" Список был закрыт до внесения последнего элемента!");
                break;
            }

            SpooledFile splf = (SpooledFile)splfList.getObject(listed++);
            if (splf != null)
            {
                // Вывод имени буферного файла
                strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                System.out.println(" буферный файл = " + strSpooledFileName);
            }
        }

    } while (!fCompleted);

    // освобождение ресурсов после заполнения списка
    splfList.close();
    splfList.removePrintObjectListListener(this);
}
```

```
    catch( ExtendedIllegalStateException e )
    {
        System.out.println(" Список был закрыт до внесения последнего элемента!");
    }

    catch( Exception e )
    {
        // ...обработка других исключительных ситуаций...
        e.printStackTrace();
    }

}

// Здесь интерактивная нить ожидает, пока она будет активизирована
// фоновой нитью при обновлении списка или завершении его создания.
private synchronized void waitForWakeUp() throws InterruptedException
{
    // Не возвращаться в состояние ожидания, если список полностью составлен
    if (!fListCompleted)
    {
        wait();
    }
}

// Следующие методы реализуют интерфейс PrintObjectListListener

// Данный метод вызывается при закрытии списка.
public void listClosed(PrintObjectListEvent event)
{
    System.out.println("*****Список был закрыт*****");
    fListClosed = true;
    synchronized(this)
    {
        // Установка флага "список составлен"
        // для активизации интерактивной нити.
        fListCompleted = true;
        notifyAll();
    }
}

// Этот метод вызывается после завершения создания списка.
public void listCompleted(PrintObjectListEvent event)
{
    System.out.println("*****Список был заполнен*****");
    synchronized (this)
    {
        // Установка флага "список составлен"
        // для активизации интерактивной нити.
        fListCompleted = true;
        notifyAll();
    }
}

// Этот метод вызывается при возникновении ошибки
// во время составления списка.
public void listErrorOccurred(PrintObjectListEvent event)
{
    System.out.println("*****Ошибка операций со списком*****");
    fListError = true;
    listException = event.getException();
    synchronized(this)
    {
        // Установка флага "список составлен"
        // для активизации интерактивной нити.
        fListCompleted = true;
        notifyAll();
```

```
        }
    }

    // Этот метод вызывается при открытии списка.
    public void listOpened(PrintObjectListEvent event)
    {
        System.out.println("*****Список был открыт*****");
        listObjectCount = 0;
    }

    // Этот метод вызывается при добавлении объекта в список.
    public void listObjectAdded(PrintObjectListEvent event)
    {
        // После добавления 25 новых объектов интерактивная нить
        // активизируется и считывает эти объекты...
        if( (++listObjectCount % 25) == 0 )
        {
            System.out.println("*****В список добавлены 25 объектов*****");
            synchronized (this)
            {
                // Активизация интерактивной нити
                notifyAll();
            }
        }
    }

    public static void main( String args[] )
    {
        NPExampleListSplfAsynch list = new NPExampleListSplfAsynch(new AS400());
        try{
            list.listSpooledFiles();
        }
        catch( Exception e )
        {
            e.printStackTrace();
        }
        System.exit(0);
    }

}
```

Пример: Асинхронное создание списка буферных файлов (без помощи обработчиков событий)

This example demonstrates how to list all the spooled files on the system asynchronously without using the PrintObjectListListener interface.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////////
//
// This example lists all spooled files on a system asynchronously without
// без применения интерфейса PrintObjectListListener. После открытия
// списка и до перехода в состояние ожидания создания списка возможно
// выполнение операций.
//
///////////////////////////////////////////////////////////////////////////
//
// Пример работы с классом PrintObjectList IBM Toolbox for 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;
    }

    // Асинхронное создание списка буферных файлов системы
    public void listSpooledFiles()
    {
        try
        {
            String strSpooledFileName;
            int listed, size;

            if( system_ == null )
            {
                system_ = new AS400();
            }

            System.out.println(
               "Получение всех буферных файлов с помощью обработчика в асинхронном режиме");

            SpooledFileList splfList = new SpooledFileList(system_);

            // Настройка фильтров - все пользователи, все очереди
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

            // Открытие списка с помощью метода openAsynchronously(),
            // возвращающего управление немедленно.
            // Обработчики событий не добавляются...
            splfList.openAsynchronously();

            System.out.println(" Выполнение операций перед ожиданием...");

            // ... выполнение операций ....

            System.out.println(" Дождитесь заполнения списка.");

            // Ожидание завершения создания списка
            splfList.waitForListToComplete();

            Enumeration enum = splfList.getObjects();

            // Вывод имен всех объектов списка
            while( enum.hasMoreElements() )
            {
                SpooledFile splf = (SpooledFile)enum.nextElement();
                if (splf != null)
                {
                    // Вывод имени буферного файла
                    strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                    System.out.println(" буферный файл = " + strSpooledFileName);
                }
            }
            // освобождение ресурсов после обработки списка
            splfList.close();
        }

        catch( Exception e )
        {
            // ...обработка исключительных ситуаций...
```

```
            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);
    }
}
```

Пример: Создание списка буферных файлов в синхронном режиме

This example demonstrates listing all spooled files on a server synchronously. Listing synchronously does not return to the caller until the complete list is built. The user perceives a slower response time as compared to listing asynchronously.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример просмотра списка буферных файлов сервера в синхронном режиме.
// При просмотре в таком режиме список файлов будет возвращен только после
// добавления в него всех объектов. Время ответа в этом случае будет больше,
// чем при просмотре списка в асинхронном режиме.
//
////////////////////////////////////////////////////////////////////////
//
// Пример работы с классом PrintObjectList IBM Toolbox for 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(" Получение списка всех буферных файлов в синхронном режиме ");
```

```
            SpooledFileList splfList = new SpooledFileList( system_ );

            // Настройка фильтров - все пользователи, все очереди
            splfList.setUserFilter("*ALL");
            splfList.setQueueFilter("/QSYS.LIB/%ALL%.LIB/%ALL%.OUTQ");

            // Открытие списка с помощью метода openSynchronously(),
            // возвращающего управление после создания списка.
            splfList.openSynchronously();
            Enumeration enum = splfList.getObjects();

            while( enum.hasMoreElements() )
            {
                SpooledFile splf = (SpooledFile)enum.nextElement();
                if ( splf != null )
                {
                    // Вывод имени буферного файла
                    strSpooledFileName = splf.getStringAttribute(SpooledFile.ATTR_SPOOLFILE);
                    System.out.println(" буферный файл = " + strSpooledFileName);
                }
            }
            // освобождение ресурсов после заполнения списка
            splfList.close();
        }
        catch( Exception e )
        {
            // ...обработка исключительных ситуаций...
            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);
    }
}
```

Пример: Применение объекта ProgramCall

This program calls the QWCRSSTS server program to retrieve the status of the system.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта ProgramCall. Эта программа вызывает
// программу  QWCRSSTS сервера для получения информации о состоянии
// системы.
//
// Формат вызова:
//    PCSystemStatusExample система
//
// Пример применения класса ProgramCall IBM Toolbox for 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( " " );



      // Если система не указана - вывод справки и завершение работы.

      if (parameters.length >= 1)
      {

         try
         {
            // Создание объекта AS400 для системы, содержащей
            // программу. Предполагается, что первый параметр - это имя системы.

            AS400 as400 = new AS400(parameters[0]);



            // Создание пути к программе.

            QSYSObjectPathName programName = new QSYSObjectPathName("QSYS", "QWCRSSTS", "PGM");



            // Создание объекта вызова программы, связанного с
// ранее созданным объектом AS400.

            ProgramCall getSystemStatus = new ProgramCall(as400);



            // Создание списка параметров программы.
// Данная программа поддерживает 5 параметров.

            ProgramParameter[] parmlist = new ProgramParameter[5];



            // Программа сервера возвращает данные в первом параметре, который
// параметром ввода-вывода. Выделение 64 байтов для этого параметра.

            parmlist[0] = new ProgramParameter( 64 );



            // Параметр 2 задает размер буфера для параметра 1. Он является входным
// параметром ввода-вывода. Ему присваивается значение 64, после чего параметр преобразуется
            // в формат сервера и добавляется в список параметров.

            AS400Bin4 bin4 = new AS400Bin4( );
            Integer iStatusLength = new Integer( 64 );
            byte[]  statusLength = bin4.toBytes( iStatusLength );
            parmlist[1] = new ProgramParameter( statusLength );



            // Параметр 3 задает формат состояния. Он является символьным
```

```
// параметром ввода-вывода. Ему присваивается строчное значение, параметр преобразуется
            // в формат сервера и добавляется в список параметров.

            AS400Text text1 = new AS400Text(8, as400);
            byte[] statusFormat = text1.toBytes("SSTS0200");
            parmlist[2] = new ProgramParameter( statusFormat );



            // Параметр 4 служит для сброса данных статистики. Он является символьным
// параметром ввода-вывода. Ему присваивается строчное значение, параметр преобразуется
            // в формат сервера и добавляется в список параметров.

            AS400Text text3 = new AS400Text(10, as400);
            byte[] resetStats = text3.toBytes("*NO       ");
            parmlist[3] = new ProgramParameter( resetStats );



            // Параметр 5 служит для передачи информации об ошибках. Он является
// параметром ввода-вывода. Параметр добавляется в список параметров.

            byte[] errorInfo = new byte[32];
            parmlist[4] = new ProgramParameter( errorInfo, 0 );



            // Указание вызываемой программы и списка параметров
            // для объекта вызова.

            getSystemStatus.setProgram(programName.getPath(), parmlist );



            // Запуск программы и переход в состояние ожидания.
            // Программа запускается дважды, так как первый набор результатов
// обычно содержит завышенные значения. Если удалить первый набор
// результатов и повторить вызов программы через пять секунд,
            // значения будут более точными.

            getSystemStatus.run();
            Thread.sleep(5000);



            // Запуск программы

            if (getSystemStatus.run()!=true)
            {

               // Если программа не запущена - получение списка сообщений
               // об ошибках из объекта программы и вывод этих сообщений.
// Наиболее вероятные ошибки: программа не найдена,
               // нет прав доступа к программе.

               AS400Message[] msgList = getSystemStatus.getMessageList();

               System.out.println("Программа не выполнена. Сообщения сервера:");

               for (int i=0; i<msgList.length; i++)
               {
                   System.out.println(msgList[i].getText());
               }
            }


            // Программа была запущена:
```

```
            else
            {

               // Создание объекта преобразования чисел сервера в формат Java.
// С помощью этого объекта в следующей части программы полученные
               // числовые данные будут преобразованы в формат Java.

               AS400Bin4 as400Int = new AS400Bin4( );



               // Получение вывода программы. Данные вывода находятся
// в массиве байт в первом параметре.

               byte[] as400Data = parmlist[0].getOutputData();



               // Значение использования CPU находится в числовом поле, начинающемся с
               // 32-го байта буфера вывода. Преобразование его из формата сервера
// в формат Java и вывод.

               Integer cpuUtil = (Integer)as400Int.toObject( as400Data, 32 );
               cpuUtil = new Integer(cpuUtil.intValue()/10);
               System.out.print("Использование CPU:  ");
               System.out.print(cpuUtil);
               System.out.println("%");



               // Значение использования DASD находится в числовом поле, начинающемся с
               // 52-го байта буфера вывода. Преобразование его из формата сервера
// в формат Java и вывод.

               Integer dasdUtil = (Integer)as400Int.toObject( as400Data, 52 );
               dasdUtil = new Integer(dasdUtil.intValue()/10000);
               System.out.print("Использование DASD: ");
               System.out.print(dasdUtil);
               System.out.println("%");



               // Значение числа заданий находится в числовом поле, начинающемся с
               // 36-го байта буфера вывода. Преобразование его из формата сервера
// в формат Java и вывод.

               Integer nj = (Integer)as400Int.toObject( as400Data, 36 );
               System.out.print("Активные задания:      ");
               System.out.println(nj);

            }

               // Выполнение программы завершено, поэтому необходимо
               // отключиться от сервера обработки команд. Вызовы программ
// и команд обрабатываются в системе одним сервером.

            as400.disconnectService(AS400.COMMAND);
         }
         catch (Exception e)
         {
            // Если какие-либо из приведенных выше операций не были выполнены -
            // появляются сообщение об ошибке и текст исключительной ситуации.

            System.out.println("Сбой вызова программы");
            System.out.println(e);
         }
```

```
      }

      // Если заданы неверные параметры, вывести текст справки.

      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
         System.out.println("   PCSystemStatusExample myServer");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println("   myServer = система, для которой будет показана информация о состоянии ");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println("   PCSystemStatusExample mySystem");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}
```

Пример: Применение классов доступа на уровне записей

This IBM Toolbox for Java example program will prompt the user for the name of the server and the file to display. The file must exist and contain records. Each record in the file will be displayed to System.out.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с файлом на уровне записей. Эта программа предложит пользователю
// указать имя сервера и просматриваемый файл. Этот файл должен существовать и
// содержать записи. Каждая запись файла будет выведена в
// System.out.
//
// Формат вызова: java RLSequentialAccessExample
//
// Пример применения класса RecordLevelAccess IBM Toolbox for Java
//
////////////////////////////////////////////////////////////////////////////



import java.io.*;
import java.util.*;
import com.ibm.as400.access.*;

public class RLSequentialAccessExample
{
   public static void main(String[] parmeters)
   {
      // Создание программы чтения ввода пользователя
            BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

      // Переменные для хранения имени системы, библиотеки, файла и элемента
      String systemName = "";
      String library = "";
      String file = "";
      String member = "";
```

```
      // Получение имени системы и файла от пользователя
      System.out.println ();

              try
      {
        System.out.print("Имя системы: ");
        systemName = inputStream.readLine();

        System.out.print("Библиотека, в которой расположен файл: ");
        library = inputStream.readLine();

        System.out.print("Имя файла: ");
        file = inputStream.readLine();

        System.out.print("Имя элемента (для чтения первого элемента нажмите Enter): ");
        member = inputStream.readLine();
        if (member.equals(""))
        {
          member = "*FIRST";
        }

        System.out.println ();

              }
      catch (Exception e)
      {
        System.out.println("Ошибка ввода пользователя.");
        e.printStackTrace();
        System.exit(0);
      }

      // Создание объекта AS400 и подключение к службе доступа на уровне записей.
      AS400 system = new AS400(systemName);
      try
      {
        system.connectService(AS400.RECORDACCESS);
      }
      catch(Exception e)
      {
        System.out.println("Не удалось создать соединение для доступа на уровне записей.");
        System.out.println("Специальные указания по созданию соединений с доступом на
                           уровне записей приведены в файле readme");
        e.printStackTrace();
        System.exit(0);
      }

      // Создание объекта QSYSObjectPathName для получения формы пути
      // к файлу в интегрированной файловой системе.
      QSYSObjectPathName filePathName = new QSYSObjectPathName(library, file, member, "MBR");

      // Создание объекта SequentialFile, представляющего просматриваемый файл
      SequentialFile theFile = new SequentialFile(system, filePathName.getPath());

      // Получение информации о формате записи для файла
      AS400FileRecordDescription recordDescription =
        new AS400FileRecordDescription(system, filePathName.getPath());
      try
      {
        RecordFormat[] format = recordDescription.retrieveRecordFormat();

        // Указание формата записи для файла
        theFile.setRecordFormat(format[0]);

        // Открытие файла для чтение. Чтение по 100 записей.
theFile.open(AS400File.READ_ONLY, 100, AS400File.COMMIT_LOCK_LEVEL_NONE);
```

```
        // Вывод каждой записи файла
        System.out.println("Просмотр файла " + library.toUpperCase() + "/"
                           + file.toUpperCase() + "(" + theFile.getMemberName().trim() + "):");

        Record record = theFile.readNext();
        while (record != null)
        {
          System.out.println(record);
          record = theFile.readNext();
        }
        System.out.println ();


        // Закрыть файл
        theFile.close();

        // Отключение от службы доступа на уровне записей
        system.disconnectService(AS400.RECORDACCESS);
      }
      catch (Exception e)
      {
        System.out.println("При попытке просмотра файла произошла ошибка.");
        e.printStackTrace();

        try
        {
          // Закрыть файл
          theFile.close();
        }
        catch(Exception x)
        {
        }

        // Отключение от службы доступа на уровне записей
        system.disconnectService(AS400.RECORDACCESS);
        System.exit(0);
      }

      // Убедитесь, что приложение завершило работу (см. файл readme)
      System.exit(0);
    }
  }
```

Пример: Применение классов доступа на уровне записей для чтения записей из файла

This program uses the record-level access classes to read records from a file on the server.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////////
//
// Пример доступа на уровне записей. Эта программа с помощью классов
// на уровне записей считывает записи из файла сервера.
//
// Формат вызова:
//    java RLReadFile сервер
//
// Эта программа считывает записи из файла примера базы данных CA/400
// (файл QCUSTCDT в библиотеке QIWS). Если вы захотите изменить этот пример
// для обновления записей в файле, создайте для работы копию файла QCUSTCDT.
//
// Пример применения класса RecordLevelAccess IBM Toolbox for 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        = "";

      // Проверка, указано ли имя системы.

      if (parameters.length >= 1)
      {

         try
         {

            // Имя системы указано в первом параметре.

            system = parameters[0];

            // Создание объекта AS400 для сервера, содержащего файл.

            AS400 as400 = new AS400(system);


            // Создание описания записей файла.
// Файл QCUSTCDT находится в библиотеке QIWS.

            ZonedDecimalFieldDescription customerNumber =
                         new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,0),
                                                          "CUSNUM");
            CharacterFieldDescription lastName =
                         new CharacterFieldDescription(new AS400Text(8, as400), "LSTNAM");

            CharacterFieldDescription initials =
                         new CharacterFieldDescription(new AS400Text(3, as400), "INIT");

            CharacterFieldDescription street =
                         new CharacterFieldDescription(new AS400Text(13, as400), "STREET");

            CharacterFieldDescription city =
                         new CharacterFieldDescription(new AS400Text(6, as400), "CITY");

            CharacterFieldDescription state =
                         new CharacterFieldDescription(new AS400Text(2, as400), "STATE");

            ZonedDecimalFieldDescription zipCode =
                         new ZonedDecimalFieldDescription(new AS400ZonedDecimal(5,0),
                                                          "ZIPCOD");
            ZonedDecimalFieldDescription creditLimit =
                         new ZonedDecimalFieldDescription(new AS400ZonedDecimal(4,0),
                                                          "CDTLMT");
            ZonedDecimalFieldDescription chargeCode =
                         new ZonedDecimalFieldDescription(new AS400ZonedDecimal(1,0),
                                                          "CHGCOD");
            ZonedDecimalFieldDescription balanceDue =
                         new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                                          "BALDUE");
            ZonedDecimalFieldDescription creditDue =
                         new ZonedDecimalFieldDescription(new AS400ZonedDecimal(6,2),
                                                          "CDTDUE");
```

```
            // Для файла DDM должно быть задано имя формата записей.
            // Имя формата записей для файла QCUSTCDT - 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);


            // Создание объекта последовательного файла, представляющего
            // файл на сервере. Преобразование имени файла в нужный формат
// с помощью объекта QSYSObectPathName.

            QSYSObjectPathName fileName = new QSYSObjectPathName("QIWS",
                                                                 "QCUSTCDT",
                                                                 "FILE");

            SequentialFile file = new SequentialFile(as400, fileName.getPath());


            // Передача формата записей объекту файла.

            file.setRecordFormat(qcustcdt);


            // Открытие файла для чтения с размером блока, равным 10 записям
// (объект файла будет получать 10 записей при каждом обращении
            // к серверу). Управление фиксацией выключено.
file.open(SequentialFile.READ_ONLY,
                      10,
                      SequentialFile.COMMIT_LOCK_LEVEL_NONE);


            // Чтение первой записи из файла.

            Record data = file.readNext();


            // Обработка всех записей файла.
while (data != null)
            {

               // Если баланс положителен, вывод имени клиента и состояния
// баланса. В следующем коде выполняется получение значений
// баланса. В следующем коде выполняется получение значений
// полей записи по их имени. При получении значение
               // преобразуется из формата сервера в формат Java.

               if (((BigDecimal)data.getField("BALDUE")).floatValue() > 0.0)
               {
                  System.out.print((String) data.getField("INIT") + "  ");
                  System.out.print((String) data.getField("LSTNAM") + "  ");
                  System.out.println((BigDecimal) data.getField("BALDUE"));
               }

               // Чтение следующей записи из файла.
```

```
            data = file.readNext();
         }


         // Отключение от сервера после обработки всех записей.

         as400.disconnectAllServices();
      }

      catch (Exception e)
      {

         // Если какие-либо из приведенных операций не были выполнены -
         // появляется сообщение об ошибке и текст исключительной ситуации.

         System.out.println("Не удалось выполнить чтение данных из файла");
         System.out.println(e);
      }
   }


   // Если заданы неверные параметры, вывести текст справки.

   else
   {
      System.out.println("");
      System.out.println("");
      System.out.println("");
      System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
      System.out.println("   RLReadFile as400");
      System.out.println("");
      System.out.println("Где");
      System.out.println("");
      System.out.println("   as400 = система, в которой расположен файл");
      System.out.println("");
      System.out.println("Например:");
      System.out.println("");
      System.out.println("   RLReadFile mySystem");
      System.out.println("");
      System.out.println("");
      System.out.println("Обратите внимание на то, что в этой программе
             выполняется чтение файла базы данных QIWS/QCUSTCDT.  ");
      System.out.println("");
      System.out.println("");
   }

   System.exit(0);

  }
}
```

Пример: Применение классов доступа к записям для чтения записей по ключу

This program uses the record-level access classes to read records by key from a file on the server. The user will be prompted for the server name to which to run and the library in which to create file QCUSTCDTKY.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример доступа на уровне записей. Эта программа с помощью классов
// доступа на уровне записей считывает записи по ключу из файла сервера.
// У пользователя будет запрошено имя сервера, в котором будет выполнена
// программа, и имя библиотеки, в которой будет создан файл QCUSTCDTKY.
//
```

```
// Формат вызова:
//    java RLKeyedFileExample
//
// This program will copy the records from the System i Access for Windows sample
// for Windows (файл QCUSTCDT в библиотеке QIWS) в файл QCUSTCDTKY, формат
// которого совпадает с форматом QIWS/QCUSTCDT, но в качестве ключа файла
// задано поле CUSNUM.
//
// Пример применения класса RecordLevelAccess IBM Toolbox for Java.
//
////////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.util.*;
import java.math.*;
import com.ibm.as400.access.*;

public class RLKeyedFileExample
{
  public static void main(String[] parameters)
  {

    // Создание программы чтения ввода пользователя.
    BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

    // Переменные для хранения имени системы, библиотеки, файла и элемента
    String systemName = "";
    String library = "";

    // Получение имени системы от пользователя
    System.out.println();
    try
    {
      System.out.print("Имя системы: ");
      systemName = inputStream.readLine();

      System.out.print("Библиотека, в которой будет создан файл QCUSTCDTKY: ");
      library = inputStream.readLine();
    }
    catch(Exception e)
    {
      System.out.println("Ошибка ввода пользователя.");
      e.printStackTrace();
      System.exit(0);
    }

    // Создание объекта AS400 и подключение к службе доступа на уровне записей.
    AS400 system = new AS400(systemName);
    try
    {
      system.connectService(AS400.RECORDACCESS);
    }
    catch(Exception e)
    {
      System.out.println("Не удалось создать соединение с доступом на уровне записей.");
      System.out.println("Специальные указания по созданию соединений с доступом на
                          уровне записей приведены в файле readme");
      e.printStackTrace();
      System.exit(0);
    }

    RecordFormat qcustcdtFormat = null;
    try
    {
      // Создание объекта RecordFormat для создания файла. Формат записей нового
// файла совпадает с форматом записей файла QIWS/QCUSTCDT. Однако,
```

```
// ключевым является поле CUSNUM будет ключевым.
      AS400FileRecordDescription recordDescription =
        new AS400FileRecordDescription(system, "/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE");

      // Для данного файла определен только один формат записей, поэтому будет
      // получен первый (и единственный) элемент массива RecordFormat,
      // который и будет использован в качестве формата записей файла.
      System.out.println("Получение формата записей файла QIWS/QCUSTCDT...");
      qcustcdtFormat = recordDescription.retrieveRecordFormat()[0];
      // Выбор CUSNUM в качестве ключевого поля
      qcustcdtFormat.addKeyFieldDescription("CUSNUM");
    }
    catch(Exception e)
    {
      System.out.println("Не удалось получить формат записей файла QIWS/QCUSTCDT");
      e.printStackTrace();
      System.exit(0);
    }

    // Создание объекта файла с доступом по ключу, который будет представлять
    // файл, создаваемый на сервере. Имя файла преобразуется в нужный формат
// с помощью объекта QSYSObectPathName.
    QSYSObjectPathName fileName = new QSYSObjectPathName(library,
                                                         "QCUSTCDTKY",
                                                         "*FILE",
                                                         "MBR");
    KeyedFile file = new KeyedFile(system, fileName.getPath());

    try
    {
      System.out.println("Создание файла " + library + "/QCUSTCDTKY...");
      // Создание файла с помощью объекта qcustcdtFormat
      file.create(qcustcdtFormat, "Файл QCUSTCDT с доступом по ключу");

      // Заполнение файла записями из QIWS/QCUSTCDT
      copyRecords(system, library);

      // Открытие файла с доступом только для чтения. Так как доступ к файлу
// accessing the file, specify a blocking factor of 1.        // Параметр фиксации уровня блокировки игнорируется, поскол
      // запущено управление фиксацией.
      file.open(AS400File.READ_ONLY,
                1,
                AS400File.COMMIT_LOCK_LEVEL_NONE);

      // Допустим, необходимо вывести информацию о заказчиках 192837, 392859 и
// 938472. Поле CUSNUM является зонным десятичным полем длинной 6
      // без дробной части. Поэтому значение ключевого поля представлено
// типом BigDecimal.
      BigDecimal[] keyValues = {new BigDecimal(192837),
                                new BigDecimal(392859),
                                new BigDecimal(938472)};

      // Создание ключа для чтения записей.
// Ключ для KeyedFile задается с помощью Object[]
      Object[] key = new Object[1];

      Record data = null;
      for (int i = 0; i < keyValues.length; i++)
      {
        // Настройка ключа для чтения
        key[0] = keyValues[i];

        // Чтение записи для заказчика номер keyValues[i]
        data = file.read(key);
        if (data != null)
        {
          // Если баланс положителен, вывод имени клиента и состояния
```

```
// баланса. В следующем коде выполняется получение значений
// баланса. В следующем коде выполняется получение значений
// полей записи по их имени. При получении значение
          // преобразуется из формата сервера в формат Java.
          if (((BigDecimal)data.getField("BALDUE")).floatValue() > 0.0)
          {
            System.out.print((String) data.getField("INIT") + "  ");
            System.out.print((String) data.getField("LSTNAM") + "  ");
            System.out.println((BigDecimal) data.getField("BALDUE"));
          }
        }
      }

      // Все операции с файлом выполнены
      file.close();

      // Удаление файла из системы пользователя
      file.delete();
    }
    catch(Exception e)
    {
      System.out.println("Не удалось создать/считать данные из файла QTEMP/QCUSTCDT");
      e.printStackTrace();
      try
      {
        file.close();
        // Удаление файла из системы пользователя
        file.delete();
      }
      catch(Exception x)
      {
      }
    }

    // Все операции с доступом на уровне записи выполнены;
    // прерывание соединения с сервером доступа на уровне записей.
    system.disconnectService(AS400.RECORDACCESS);
    System.exit(0);
  }

  public static void copyRecords(AS400 system, String library)
  {
    // Запуск с помощью класса CommandCall команды CPYF для копирования записей
    // из QIWS/QCUSTCDT в QTEMP/QCUSTCDT
    CommandCall c = new CommandCall(system, "CPYF FROMFILE(QIWS/QCUSTCDT) TOFILE("
                                    + library + "/QCUSTCDTKY) MBROPT(*REPLACE)");
    try
    {
      System.out.println("Копирование записей файла QIWS/QCUSTCDT в файл "
                         + library + "/QCUSTCDTKY...");
      c.run();
      AS400Message[] msgs = c.getMessageList();
      if (!msgs[0].getID().equals("CPC2955"))
      {
        System.out.println("Не удалось заполнить файл " + library + "/QCUSTCDTKY");
        for (int i = 0; i < msgs.length; i++)
        {
          System.out.println(msgs[i]);
        }
        System.exit(0);
      }
    }
    catch(Exception e)
    {
      System.out.println("Не удалось заполнить файл " + library + "/QCUSTCDTKY");
```

```
      System.exit(0);
    }
  }
}
```

Пример: Применение класса UserList для получения списка пользователей указанной группы

This source is an example of IBM Toolbox for Java UserList. This program lists all of the users in a given group.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
//////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта UserList. Данная программа
// показывает всех пользователей указанной группы.
//
// Формат вызова:
//    UserListExample система группа
//
// Пример применения класса UserList IBM Toolbox for Java.
//
//////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import java.util.Enumeration;

public class UserListExample
{


    public static void main(String[] args)
    {
        // Если система и группа не указаны -
        // вывод справки и завершение работы.
        if (args.length != 2)
        {
            System.out.println("Применение: UserListExample система группа");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Имя группы передается во втором аргументе командной строки.
String groupName = args[1];

            // Создание объекта списка пользователей.
            UserList userList = new UserList (system);

            // Получение списка пользователей указанной группы.
            userList.setUserInfo (UserList.MEMBER);
            userList.setGroupInfo (groupName);
            Enumeration enum = userList.getUsers ();

            // Вывод в цикле
            // имен и описаний пользователей.
            while (enum.hasMoreElements ())
            {
                User u = (User) enum.nextElement ();
                System.out.println ("Имя пользователя:   " + u.getName ());
                System.out.println ("Описание: " + u.getDescription ());
```

```
                System.out.println ("");
            }

        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
        }

        System.exit (0);
    }


}
```

Примеры: JavaBeans

This section lists the code examples that are provided throughout the IBM Toolbox for Java bean information.

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> **Отказ от гарантий на предоставляемый код примеров**
>
> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.
>
> Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Код компонента IBM Toolbox for Java

В следующем примере создаются объекты AS400 и CommandCall и для них настраиваются обработчики событий. Обработчики выводят сообщения при подключении и отключении сервера и по окончании выполнения команды объектом CommandCall.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример использования компонентов. В программе используется
// поддержка JavaBean, предусмотренная в классах IBM Toolbox for Java.
//
// Формат вызова:
//       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_ );
```

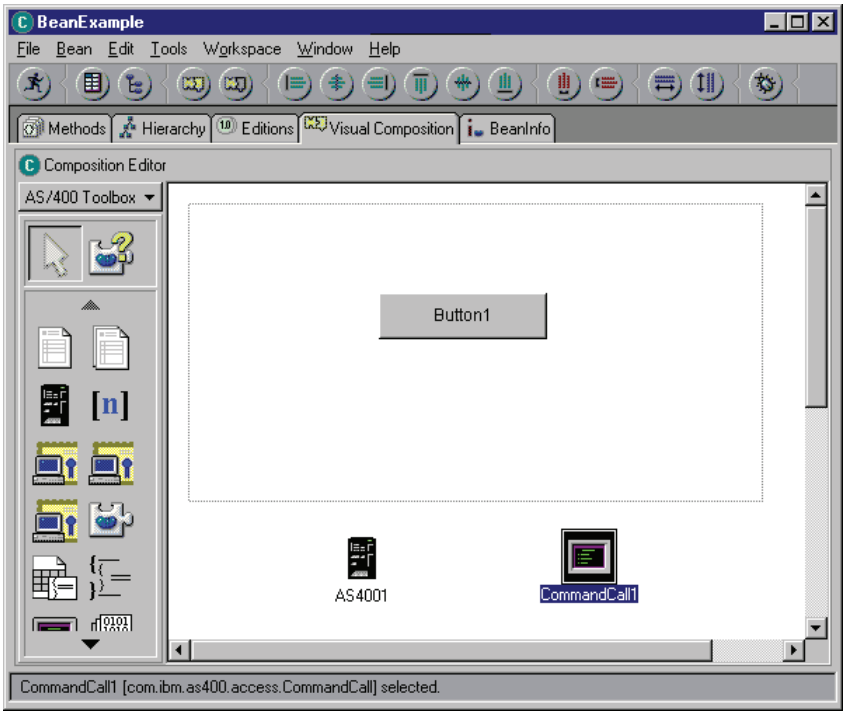
```
    BeanExample()
    {
        // Вывод сообщения при каждом подключении и отключении
        // системы с помощью обработчика событий объекта AS400.
// Объект AS400 будет вызывать этот код при каждом
        // подключении и отключении.

        as400_.addConnectionListener
        (new ConnectionListener()
          {
             public void connected(ConnectionEvent event)
             {
                 System.out.println( "Система подключена." );
}
             public void disconnected(ConnectionEvent event)
             {
                 System.out.println( "Система отключена." );
}
          }
        );

        // Вывод сообщения по завершении работы каждой команды
// с помощью обработчика событий объекта commandCall.
// Объект будет вызывать этот код при каждом запуске команды.

        cmd_.addActionCompletedListener(
            new ActionCompletedListener()
            {
                public void actionCompleted(ActionCompletedEvent event)
                {
                    System.out.println( "Команда выполнена." );
}
            }
        );
    }

    void runCommand()
    {
        try
        {
            // Выполнение команды. Обработчики событий выведут
// информацию о подключении к AS/400 и о завершении
            // работы команды.
            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);
    }
}
```

Пример: Создание объектов JavaBean с помощью визуального компоновщика

В этом примере используется IBM VisualAge for Java Enterprise Edition V2.0 Composition Editor, однако другие компоновщики отличаются от него незначительно. This example creates an applet for a button that, when pressed, runs a command on the server.

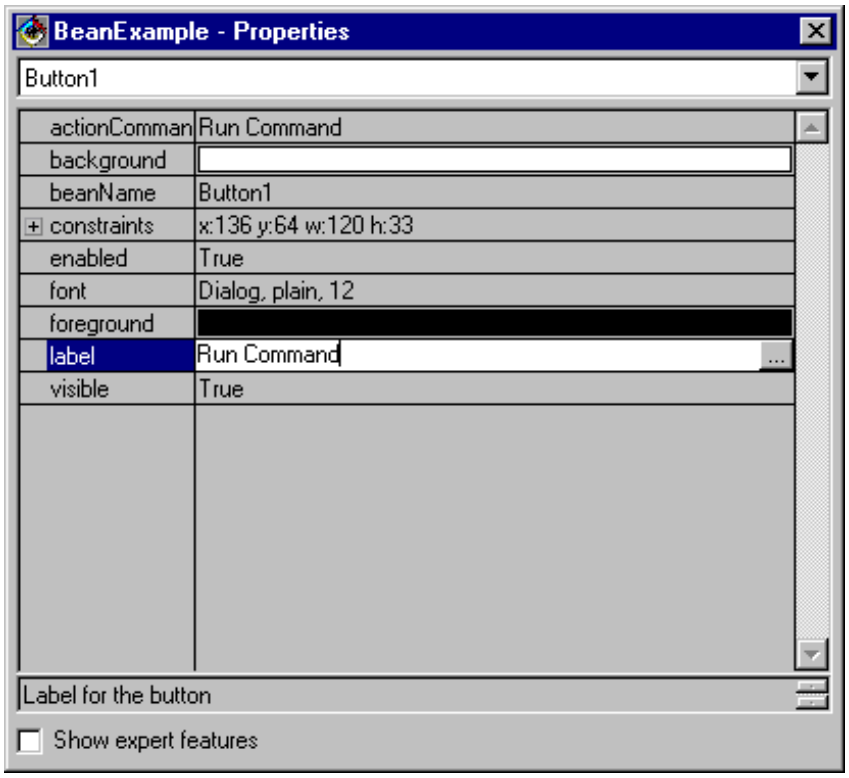
- Перенесите кнопку в окно аплета. (Кнопка находится в левой части окна визуального компоновщика, как показано на рисунке 1.)
- Перенесите компоненты CommandCall и AS400 за пределы окна аплета. (Компоненты находятся в левой части окна визуального компоновщика, как показано на рисунке 1.)

Рисунок 1: Окно визуального компоновщика VisualAge - gui.BeanExample



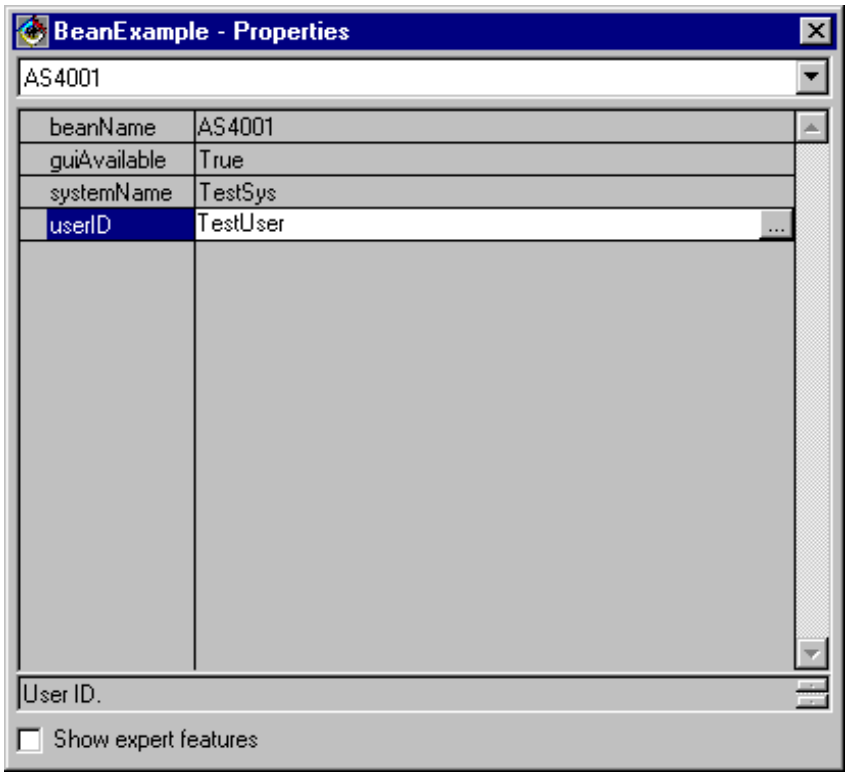
- Отредактируйте свойства компонента. (Выберите компонент и нажмите правую кнопку мыши, затем в выпадающем окне выберите опцию Свойства.)
  - Измените название кнопки на **Запустить команду**, как показано на рисунке 2.

    Рисунок 2: Изменение названия кнопки на "Запустить команду"



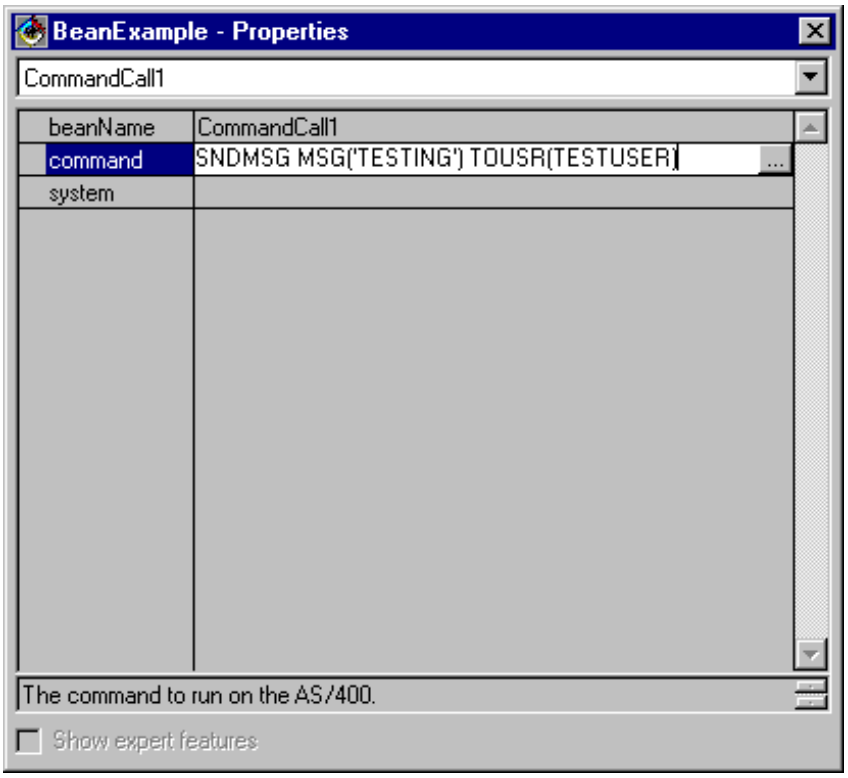
– Измените имя системы в компоненте AS400 на **TestSys**
– Измените ИД пользователя в компоненте AS400 на **TestUser**, как показано на рисунке 3.

Рисунок 3: Изменение ИД пользователя на TestUser



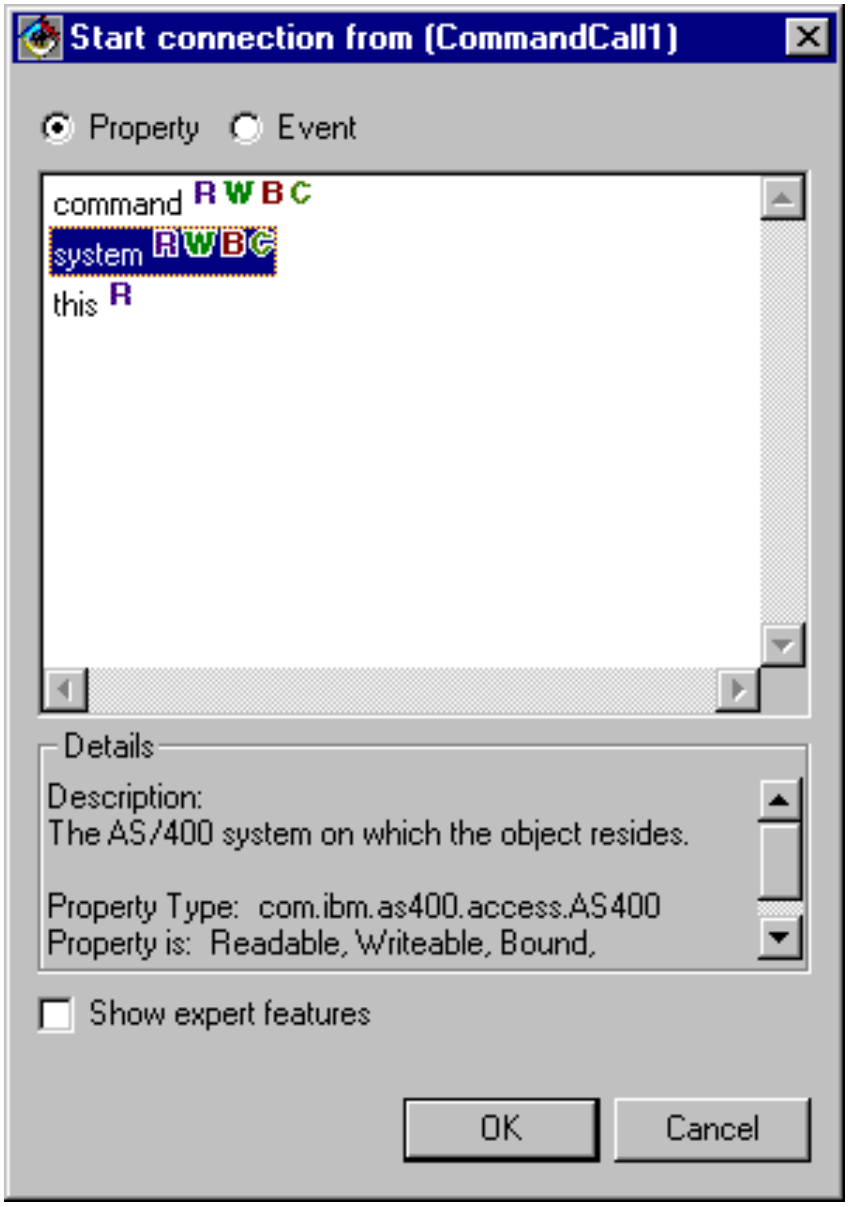
– Измените текст команды в компоненте CommandCall на **SNDMSG MSG('Проверка') TOUSR('TESTUSER')**, как показано на рисунке 4.

Рисунок 4: Изменение команды в компоненте CommandCall



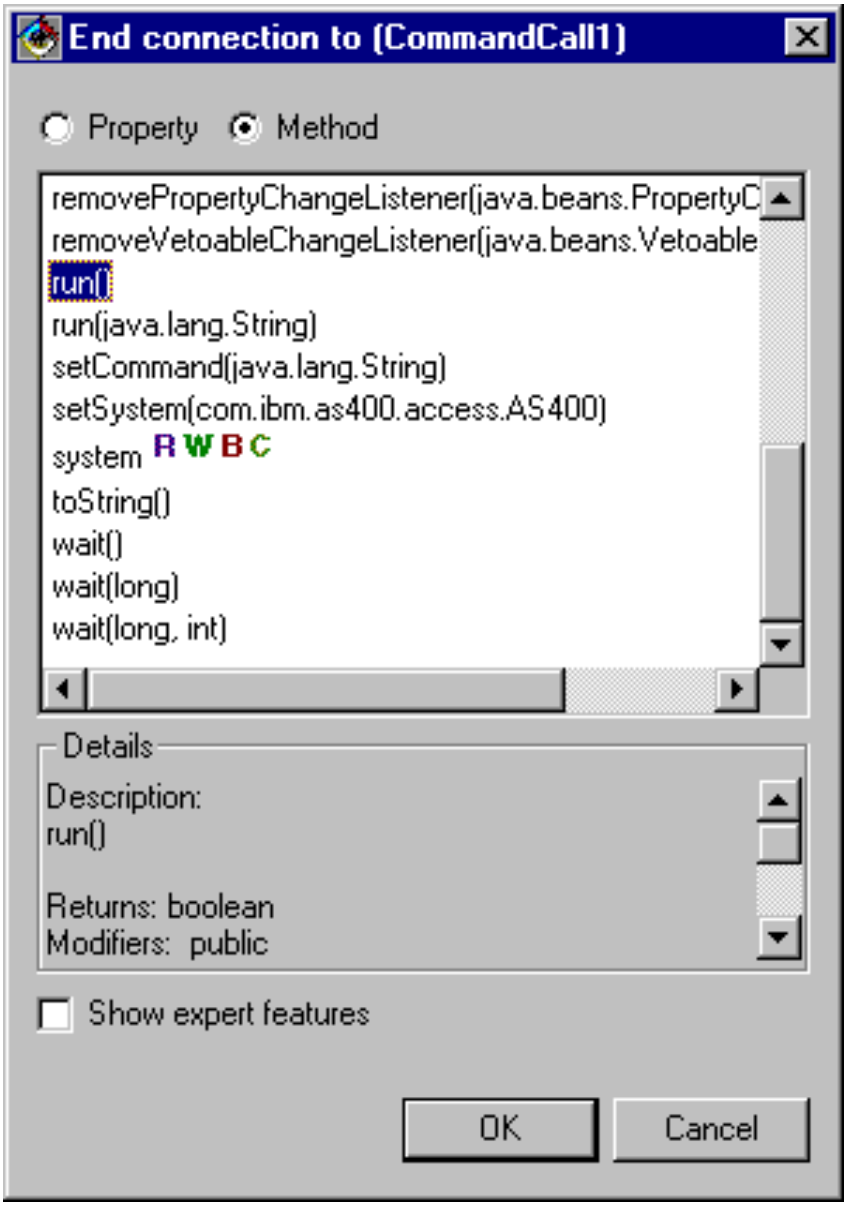
- Свяжите компонент AS400 с компонентом CommandCall. Способ связывания может быть различным в разных компоновщиках. В данном примере сделайте следующее:
  - Выберите компонент CommandCall и нажмите правую кнопку мыши
  - Выберите **Связать**
  - Выберите **Опции связывания**
  - Выберите **system** в списке опций, как показано на рисунке 5.
  - Выберите компонент AS400
  - Выберите **this** из выпадающего меню, появившегося для компонента AS400

  Рисунок 5: Связывание компонента AS400 с компонентом CommandCall



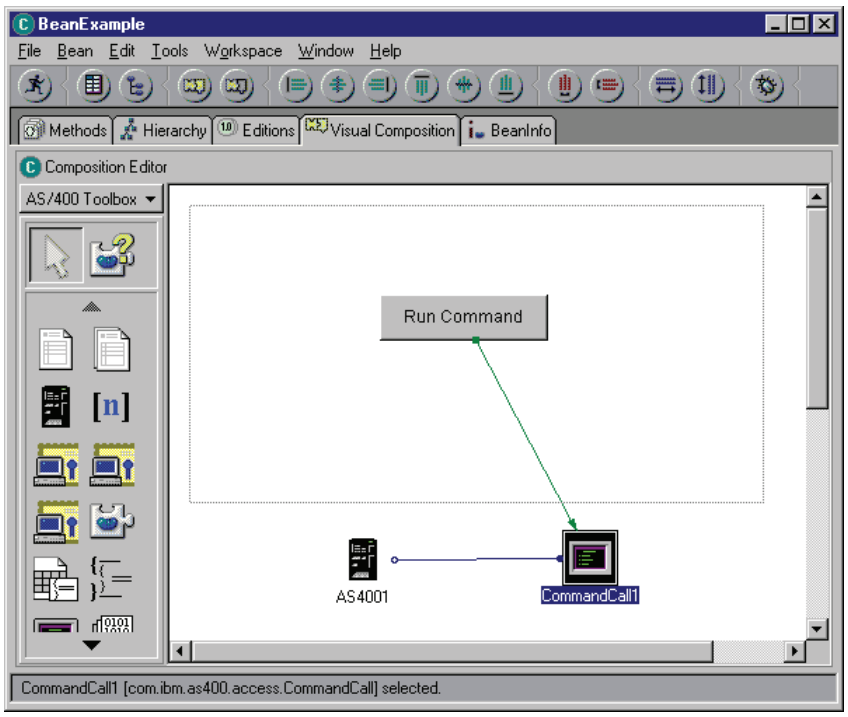
- Свяжите кнопку с компонентом CommandCall.
  - Выберите компонент Кнопка и нажмите правую кнопку мыши
  - Выберите **Связать**
  - Выберите **actionPerformed**
  - Выберите компонент CommandCall
  - Выберите **Connectable Features** из появившегося выпадающего меню
  - Выберите **run()** в списке методов, как показано на рисунке 6.

    Рисунок 6: Связывание метода с кнопкой



После выполнения описанных действий окно визуального компоновщика VisualAge должно выглядеть так, как показано на рисунке 7.

Рисунок 7: Окно визуального компоновщика VisualAge - завершение работы с примером



Примеры: Классы трассировки соединений

This topic links to the code example provided in the documentation of IBM Toolbox for Java commtrace classes.

- “Пример: Применение классов Commtrace” на стр. 191

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Примеры работы с Graphical Toolbox

Ниже приведены примеры применения графических инструментов для создания пользовательского интерфейса в приложениях.

- Создание и просмотр панели: В этом разделе описано создание простой панели, наглядно демонстрирующее основные функции и возможности среды Graphical Toolbox
- Создание и просмотр панели: Создание и просмотр панели в случае, если файлы свойств и панели расположены в одном каталоге
- Создание полнофункционального окна диалога: Рассмотрено создание полнофункционального окна диалога (для этого необходимо создать реализацию DataBean, содержащую данные панели, и идентифицировать атрибуты в файле PDML)

- Задание размеров панели с помощью динамического диспетчера панелей: В данном разделе описано изменение размеров панели с помощью динамического диспетчера панелей во время выполнения программы
- Изменяемое поле со списком: В этом разделе описано создание компонента данных для изменяемого поля со списком

В следующих примерах показано, как с помощью GUI Builder можно создавать перечисленные ниже элементы графического интерфейса:

- Панели: Создание простой панели и кода компонента данных для этой панели
- Составные панели: Создание составной панели и различные типы составных панелей
- Окна свойств: Создание окна свойств и различные типы окон свойств
- Разделенные панели: Создание разделенной панели и различные виды таких панелей
- Панели с закладками: Создание панели с закладками и различные типы панелей с закладками
- Мастеры: Создание мастера и различные типы мастеров
- Панели инструментов: Создание панели инструментов и различные типы панелей инструментов
- Строки меню: Создание строк меню и различные типы строк меню
- Справка: Создание документа справки и разбиение его на разделы. Кроме того, дополнительная информация приведена в разделе Изменение файлов справки, созданных GUI Builder
- Пример: Демонстрирует интерфейс программы PDML, включающий определения панелей, окна свойств, мастеры, переключатели и пункты меню.

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Пример: Создание панели с помощью GUI Builder

Ниже приведен пример создания простой панели с помощью набора Graphical Toolbox. В нем демонстрируются основные возможности и функции этого набора.

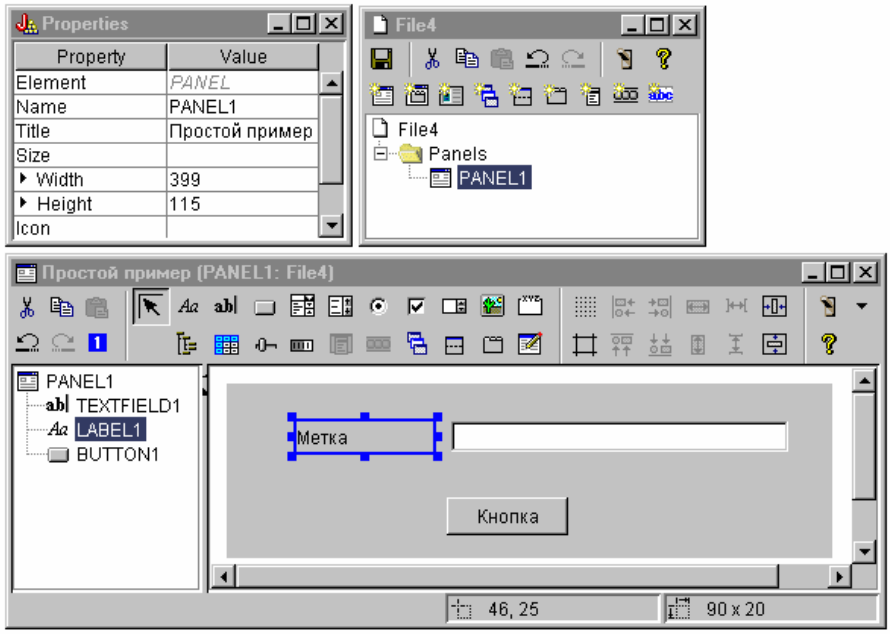
Кроме того, приводится пример небольшого приложения на Java, с помощью которого можно просмотреть созданную панель. В этом примере пользователь вводит данные в текстовом поле и нажимает кнопку **Закрыть**. После этого приложение копирует данные на консоль Java.

Создание панели

При вызове GUI Builder появляются два окна: Свойства и Редактор файлов. Создайте файл с именем ″MyGUI.pdml″. Создайте новую панель. Для этого выберите пункт ″Вставить панель″ в окне Редактор файлов. Будет создана панель ″PANEL1″. Измените ее заголовок, введя текст ″Простой пример″ в поле ″Заголовок″ окна Свойства. Удалите три кнопки, расположенные на панели. Для этого выделите их с помощью мыши и нажмите клавишу ″Delete″. Добавьте метку, текстовое поле и кнопку с помощью Редактора панелей, расположив их на панели, как показано на рис. 1.

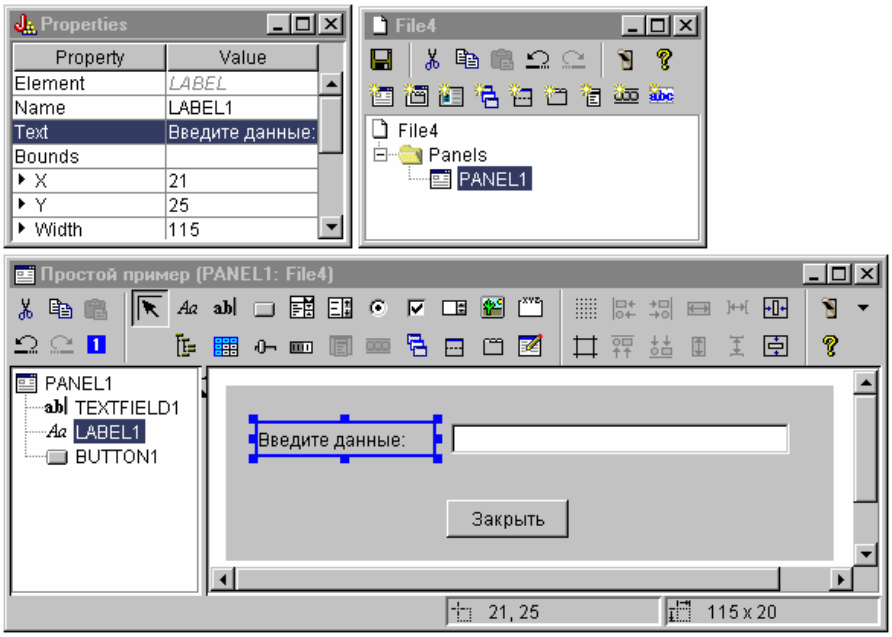
Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Рис. 1: Окна GUI Builder: Создание панели



Выделите метку и измените ее имя в окне Свойства. В этом примере название кнопки было изменено на ″Закрыть″.

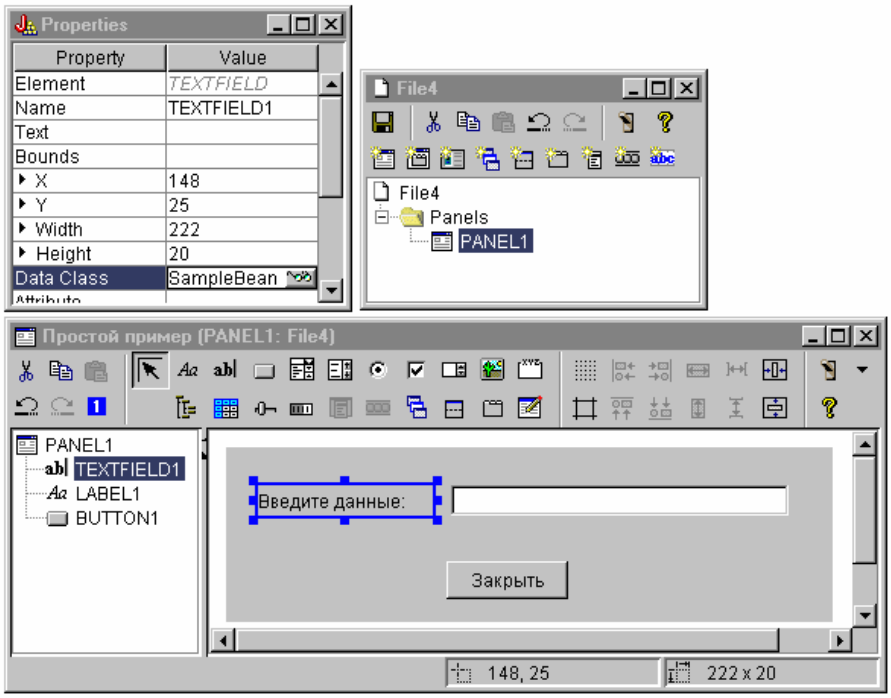
Рисунок 2: Окна GUI Builder: Изменение текста в окне свойств



Текстовое поле

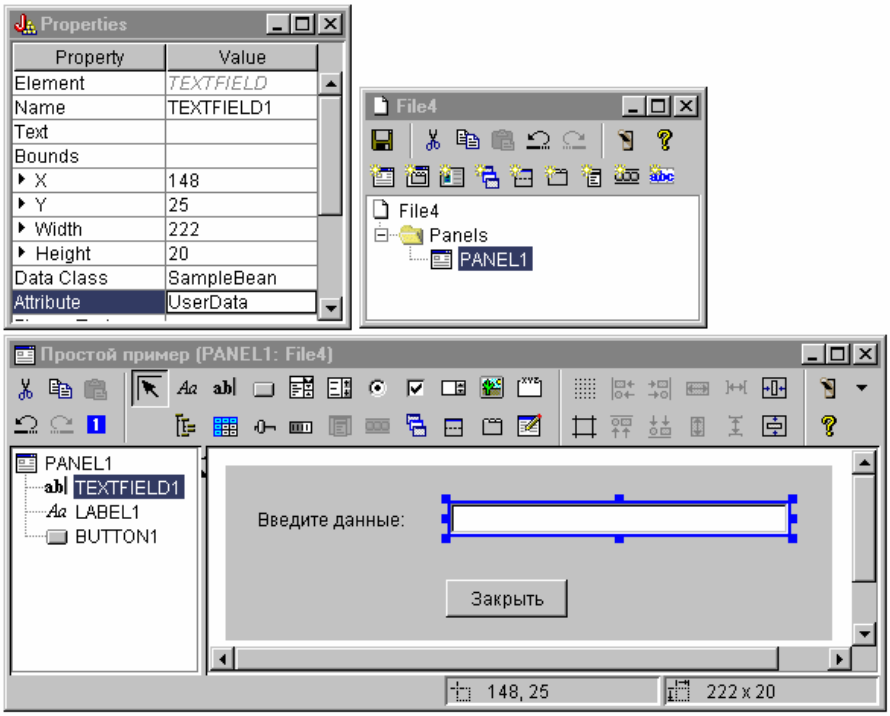
Текстовое поле будет содержать данные. Для того чтобы они были обработаны GUI Builder, измените некоторые свойства этого поля. В поле Класс данных укажите имя класса компонента **SampleBean**. Этот компонент будет содержать данные для текстового поля.

Рисунок 3: Окна GUI Builder: Установка свойства Класс данных



В поле Атрибут укажите свойство компонента, содержащее данные. В данном случае это **UserData**.

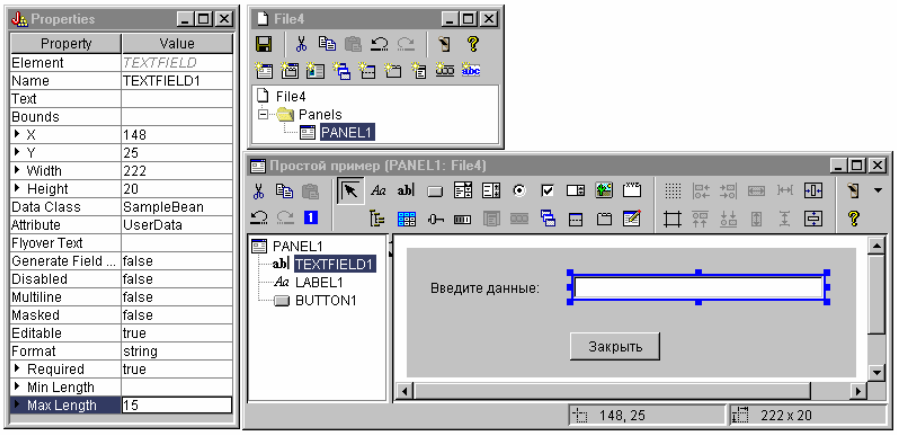
Рисунок 4: Окна GUI Builder: Установка свойства Атрибут



В результате с текстовым полем будет связано свойство **UserData**. После запуска программа Graphical Toolbox определяет начальное значение этого поля путем вызова метода **SampleBean.getUserData**. При закрытии панели указанному свойству компонента присваивается текущее значение, введенное в поле, путем вызова метода **SampleBean.setUserData**.

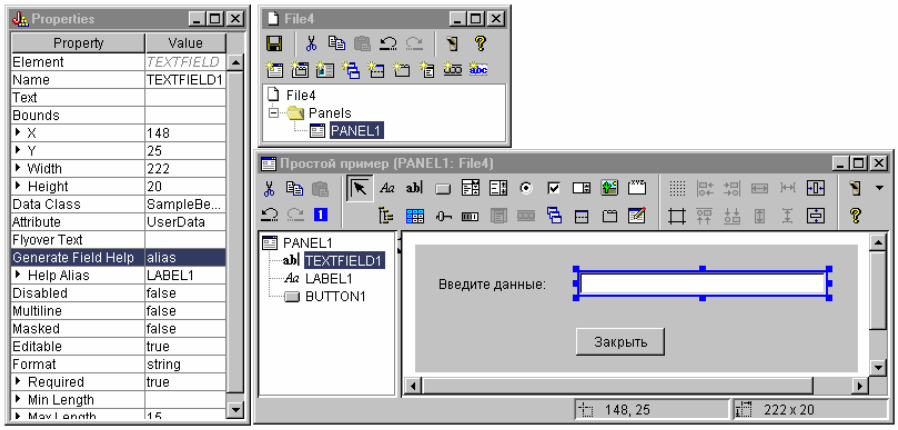
Укажите, что пользователь должен ввести строку, содержащую не больше 15 символов.

Рисунок 5: Окна GUI Builder: Установка максимальной длины текстового поля



Укажите, что контекстная справка по текстовому полю совпадает с разделом справки, связанным с меткой ″Введите данные″.

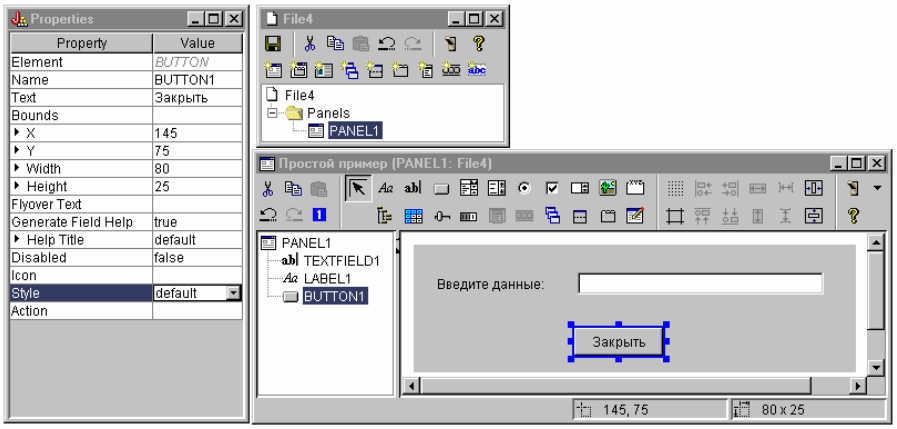
Рисунок 6: Окна GUI Builder: Задание контекстной справки для текстового поля



Кнопка

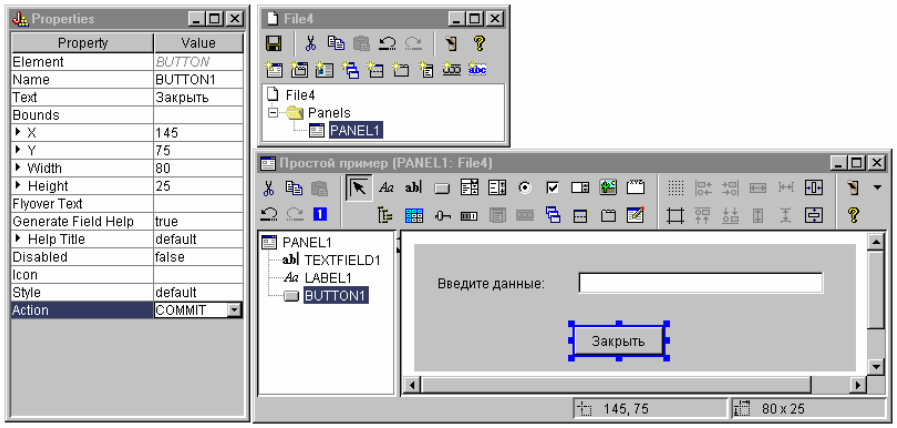
Измените свойство, задающее стиль, выбрав выделение по умолчанию.

Рисунок 7: Окна GUI Builder : Установка выделения кнопок по умолчанию с помощью свойства Стиль



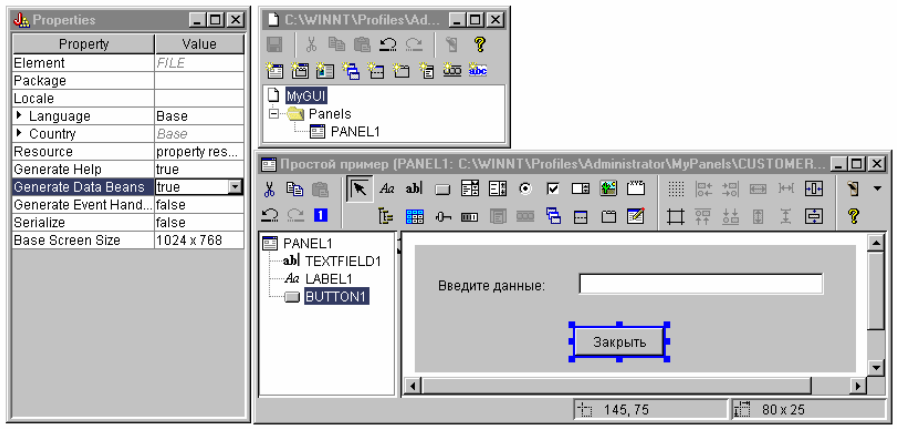
Присвойте свойству Действие значение COMMIT - тогда при нажатии кнопки будет вызываться метод компонента `setUserData`.

Рисунок 8: Окна GUI Builder: Присвоение свойству Действие значения COMMIT



Перед сохранением панели задайте свойства файла PDML, чтобы вместе с компонентом Javabean был создан шаблон справки. Сохраните файл, щелкнув на значке  в окне GUI Builder.Присвойте файлу имя **MyGUI.pdml**.

Figure 9: GUI Builder windows: Setting properties to generate the online help skeleton and the Java bean



Созданные файлы

Сохранив определение панели, просмотрите файлы, созданные GUI Builder. **PDML file** Here is the content of **MyGUI.pdml** to give you an idea of how the Panel Definition Markup Language works. Because you use PDML only through the tools provided by the Graphical Toolbox, it is not necessary to understand the format of this file in detail:

```
<!-- Generated by GUI Builder -->
<PDML version="2.0" source="JAVA" basescreensize="1280x1024">

 <PANEL name="PANEL1">
  <TITLE>PANEL1<TITLE>
  <SIZE>351,162<</SIZE>
  <LABEL name="LABEL1"">
   <TITLE>PANEL1.LABEL1</TITLE>
   <LOCATION>18,36</LOCATION>
   <SIZE>94,18</SIZE>
   <HELPLINK>PANEL1.LABEL1</HELPLINK>
  </LABEL>
  <TEXTFIELD name="TEXTFIELD1">
   <TITLE>PANEL1.TEXTFIELD1</TITLE>
   <LOCATION>125,31</LOCATION>
   <SIZE>191,26</SIZE>
   <DATACLASS>SampleBean</DATACLASS>
   <ATTRIBUTE>UserData</ATTRIBUTE>
   <STRING minlength="0" maxlength="15"/>
   <HELPALIAS>LABEL1</HELPALIAS>
  </TEXTFIELD>
  <BUTTON name="BUTTON1">
   <TITLE>PANEL1.BUTTON1</TITLE>
   <LOCATION>125,100</LOCATION>
   <SIZE>100,26</SIZE>
   <STYLE>DEFAULT<</STYLE>
   <ACTION>COMMIT</ACTION>
   <HELPLINK>PANEL1.BUTTON1</HELPLINK>
  </BUTTON>
 </PANEL>

</PDML>
```

Комплект ресурсов

С каждым файлом PDML связан комплект ресурсов. В данном примере все ресурсы, которые могут быть переведены на национальный язык, сохранены в файле свойств **MyGUI.properties**. Обратите внимание, что помимо этого файл свойств содержит параметры настройки GUI Builder.

```
##Generated by GUI Builder
BUTTON_1=Закрыть
TEXT_1=
@GenerateHelp=1
@Serialize=0
@GenerateBeans=1
LABEL_1=Введите данные:
PANEL_1.Margins=18,18,18,18,18,18
PANEL_1=Простой пример
```

JavaBean

Помимо описанных файлов, создается и файл на Java с шаблоном исходного кода объекта JavaBean. Ниже приведено содержимое файла **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 ="";
        }
}
```

Обратите внимание, что в этом шаблоне уже реализованы методы gettor и settor для свойства `UserData`. Другие методы определены в интерфейсе `DataBean`, поэтому их необходимо реализовать самостоятельно.

The GUI Builder has already invoked the Java compiler for the skeleton and produced the corresponding class file. For the purposes of this simple example, you do not need to modify the bean implementation. In a real Java application you would typically modify the `load` and `save` methods to transfer data from an external data source. The default implementation of the other two methods is often sufficient. For more information, see the documentation on the **DataBean** interface in the Javadocs for the PDML runtime framework.

Файл справки

GUI Builder создает шаблон справки в формате HTML. В нем вы можете ввести собственную справочную информацию. Дополнительная информация по этому вопросу приведена в следующих разделах:

- Создание справки
- Изменение файлов справки, созданных GUI Builder

Создание приложения

После сохранения определения панели и автоматически созданных файлов вы можете создать приложение. All you need is a new Java source file that will contain the main entry point for the application. For this example, the file is called **SampleApplication.java**. It contains the following code:

```
import com.ibm.as400.ui.framework.java.*;
import java.awt.Frame;

public class SampleApplication
{
    public static void main(String[] args)
```

```
    {
        // Создание объекта компонента, содержащего данные для панели
        SampleBean bean = new SampleBean();

        // Инициализация объекта
        bean.load();

        // Настройка DataBean для передачи компонента администратору панели
        DataBean[] beans = { bean };

        // Создание диспетчера панелей. Параметры:
        // 1. Имя файла PDML
        // 2. Имя панели
        // 3. Список объектов, содержащих данные для панели
        // 4. Фрейм AWT,
необходимый для создания модальной панели


        PanelManager pm = null;
        try { pm = new
PanelManager("MyGUI", "PANEL_1", beans, new Frame()); }
                catch (DisplayManagerException e)
        {
            // Произошла ошибка; вывод сообщения и выход из программы
            e.displayUserMessage(null);
            System.exit(1);
        }

        // Вывод панели и
передача управления
                pm.setVisible(true);

        // Копирование
сохраненных пользовательских данных в стандартный вывод
                System.out.println("ПОЛЬЗОВАТЕЛЬСКИЕ ДАННЫЕ: '" +
bean.getUserData() + "'");


        // Завершение работы
приложения
                System.exit(0);
    }
}
```

It is the responsibility of the calling program to initialize the bean object or objects by calling **load**. If the data for a panel is supplied by multiple bean objects, then each of the objects must be initialized before passing them to the Graphical Toolbox environment.

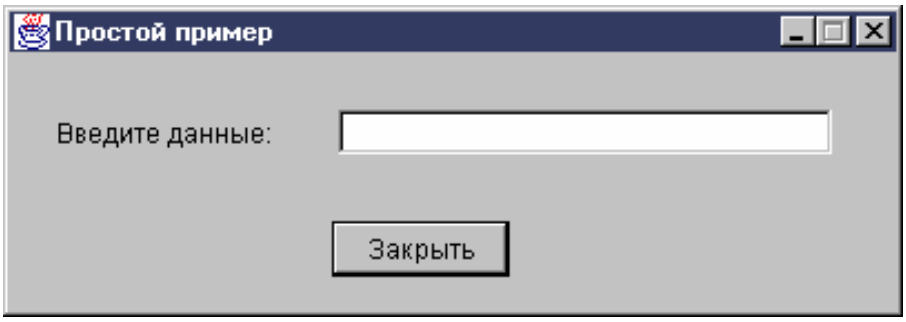
The class **com.ibm.as400.ui.framework.java.PanelManager** supplies the API for displaying standalone windows and dialogs. The name of the PDML file as supplied on the constructor is treated as a resource name by the Graphical Toolbox - the directory, ZIP file, or JAR file containing the PDML must be identified in the classpath.

Because a **Frame** object is supplied on the constructor, the window will behave as a modal dialog. In a real Java application, this object might be obtained from a suitable parent window for the dialog. Because the window is modal, control does not return to the application until the user closes the window. At that point, the application simply echoes the modified user data and exits.

Запуск приложения

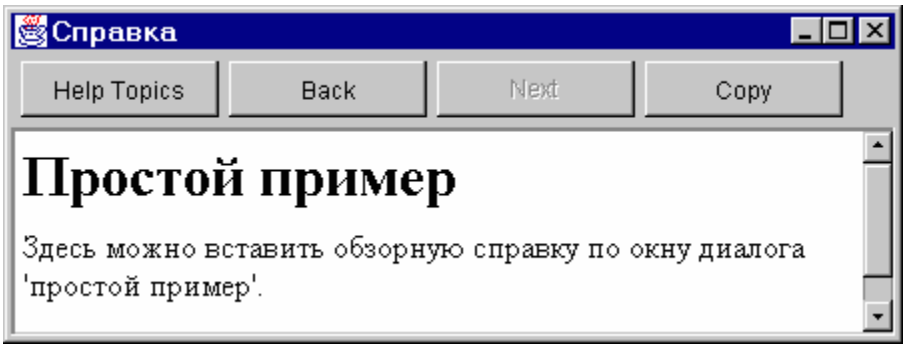
Ниже приведен примерный вид окна, которое будет открыто при запуске приложения:

Рисунок 10: Окно примера простого приложения



Если пользователь выделит текстовое поле и нажмет клавишу F1, то Graphical Toolbox откроет окно, содержащее шаблон электронной справки, созданный GUI Builder.

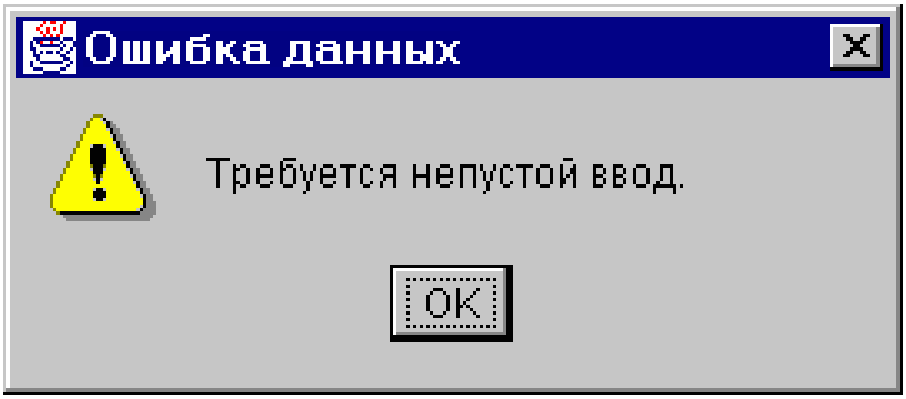
Рисунок 11: Структура электронной справки простого примера



Вы можете отредактировать документ HTML, добавив текст справки для показанных разделов.

Если в поле указаны неверные данные (например, если пользователь нажал кнопку **Закрыть**, не указав значение), Graphical Toolbox выдаст сообщение об ошибке и вновь активизирует текстовое поле для ввода данных.

Рисунок 12: Сообщение об ошибке данных



Информация о том, как запустить аплет на основе этого примера, приведена в разделе Работа с Graphical Toolbox в браузере.

Информация, связанная с данной

Package com.ibm.as400.ui.framework.java summary

Поле ввода со списком

Когда программа создания компонентов JavaBean определяет методы доступа gettor и settor для поля ввода со списком, по умолчанию она считывает строку из gettor и возвращает строку в settor. Возможно, будет полезно изменить описания так, чтобы метод settor получал в качестве аргумента класс объекта, а метод gettor возвращал тип объекта. Это позволит определить, что выбрал пользователь, с помощью ChoiceDescriptor.

Если для методов доступа установлен тип Object, система будет требовать ChoiceDescriptor или значение типа Object вместо строки.

В приведенном ниже примере предполагается, что Editable - это редактируемый объект ComboBox, значение которого равно Double, системному значению или не задано.

```
public Object getEditable()
{
    if (m_setting == SYSTEMVALUE)
    {
       return new ChoiceDescriptor("choice1","Системное значение");
    }
    else if (m_setting == NOTSET)
    {
       return new ChoiceDescriptor("choice2","Значение не задано");
    }
    else
    {
       return m_doubleValue;
    }
}
```

Аналогично, если для методов доступа установлен тип Object, система будет возвращать объект типа ChoiceDescriptor, описывающий выбранный вариант, или значение типа 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
         { /* обработка ошибок */ }
     }
```

Пример: Применение RecordListFormPane

This example program presents a form that contains the contents of a file on the server.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта RecordListFormPane.
// Данная программа показывает форму с содержимым файла сервера.
//
// Формат вызова:
//    RecordListFormPaneExample система имя-файла
//
// Пример применения класса RecordListFormPane IBM Toolbox for Java.
```

```
//
//////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class RecordListFormPaneExample
{


    public static void main(String[] args)
    {
        // Если система и имя файла не указаны -
        // вывод справки и завершение работы.
        if (args.length != 2)
        {
            System.out.println("Применение: RecordListFormPaneExample система имя-файла");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание фрейма.
            JFrame f = new JFrame ("Пример применения класса RecordListFormPane");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели формы списка записей для вывода содержимого
            // базы данных. Обратите внимание, что сначала создается панель
// и добавляется обработчик ошибок, а затем будет задается имя
            // файла и системы. Эти операции можно было бы выполнить
            // одновременно следующим образом:
            //    RecordListFormPane formPane = new RecordListFormPane (system, args[1]);
            // Потенциальная неполадка состоит в том, что обработчик ошибок
            // еще не создан, поэтому в случае ошибки в имени файла не будет
            // показано сообщение об ошибке.
            RecordListFormPane formPane = new RecordListFormPane();
            formPane.addErrorListener (errorHandler);
            formPane.setSystem(system);
            formPane.setFileName(args[1]);

            // Получение информации из системы.
            formPane.load ();

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("В центре", formPane);
            f.pack ();
            f.show ();
```

```
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Создание панели с помощью GUI Builder

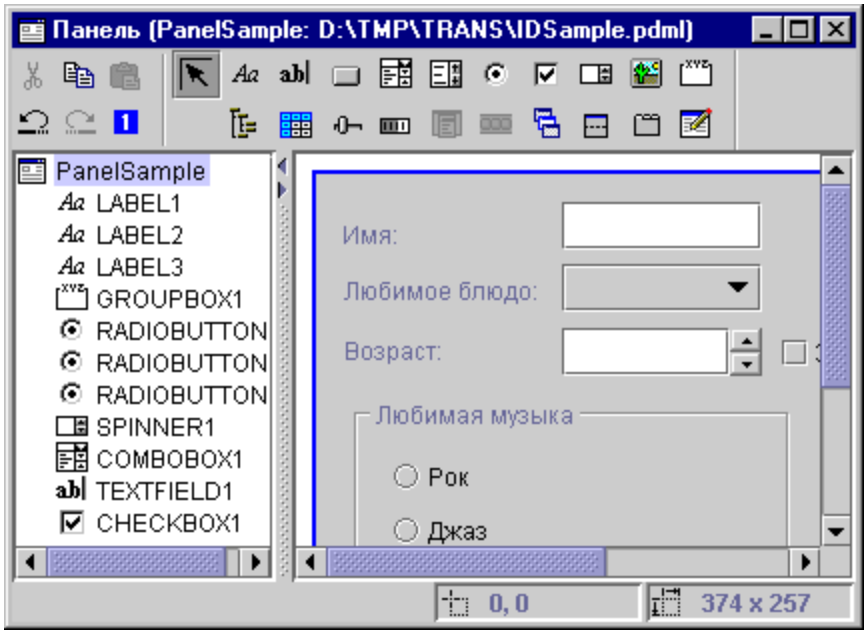
Follow these instructions to create a panel using GUI Builder.

To create a menu, from the menu bar on the main GUI Builder window, select **File → New File**.

В меню **Файл** GUI Builder щелкните на значке Вставить новую панель  , чтобы создать новую панель с помощью соответствующего инструмента. Кнопки панели инструментов в окне **Панель** соответствуют различным компонентам, которые можно добавить в панель. Выберите компонент, а затем щелкните мышью там, куда вы хотите его поместить.

Ниже приведен пример панели, созданной с помощью доступных компонентов.

Figure 1: Creating a sample Panel with GUI Builder



Панель и ее компоненты, показанные на рис. 1, описывает следующий код DataBean:

```
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("Имя = " + m_sName);
    System.out.println("Любимое блюдо = " + m_oFavoriteFood);
    System.out.println("Возраст = " + m_oAge);
    String sMusic = "";
    if (m_sFavoriteMusic != null)
    {
        if (m_sFavoriteMusic.equals("RADIOBUTTON1"))
            sMusic = "Рок";
        else if (m_sFavoriteMusic.equals("RADIOBUTTON2"))
            sMusic = "Джаз";
        else if (m_sFavoriteMusic.equals("RADIOBUTTON3"))
            sMusic = "Кантри";
```

```
        }
        System.out.println("Любимая музыка = " + sMusic);
    }

    public void load()
    {
        m_sName = "Образец имени";
        m_oFavoriteFood = null;
        m_cdFavoriteFood = new ChoiceDescriptor[0];
        m_oAge = new Integer(50);
        m_sFavoriteMusic = "RADIOBUTTON1";
    }
}
```

Панель - это один из самых простых компонентов интерфейса, которые можно создать с помощью GUI Builder. Однако даже на основе панели можно создать сложное приложение с удобным пользовательским интерфейсом.

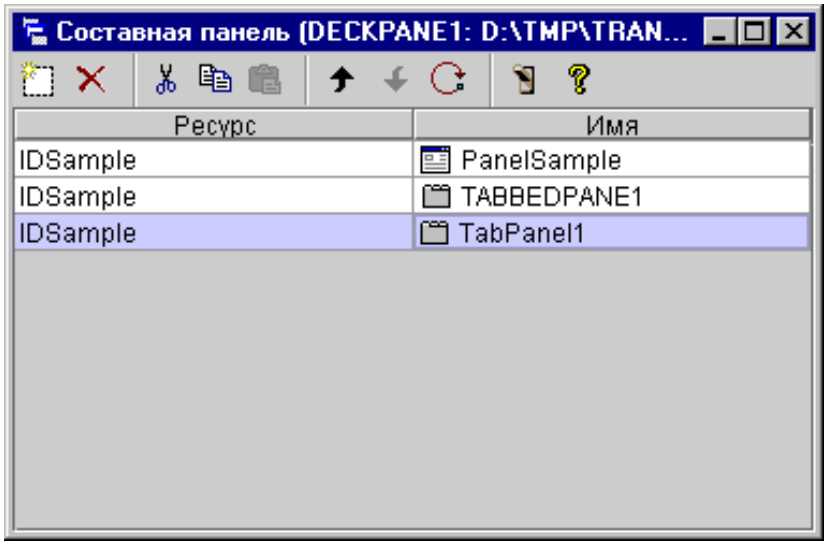
Создание составной панели с помощью GUI Builder

GUI Builder позволяет быстро создать составную панель.

From the menu bar on the GUI Builder window, select **File → New File**.

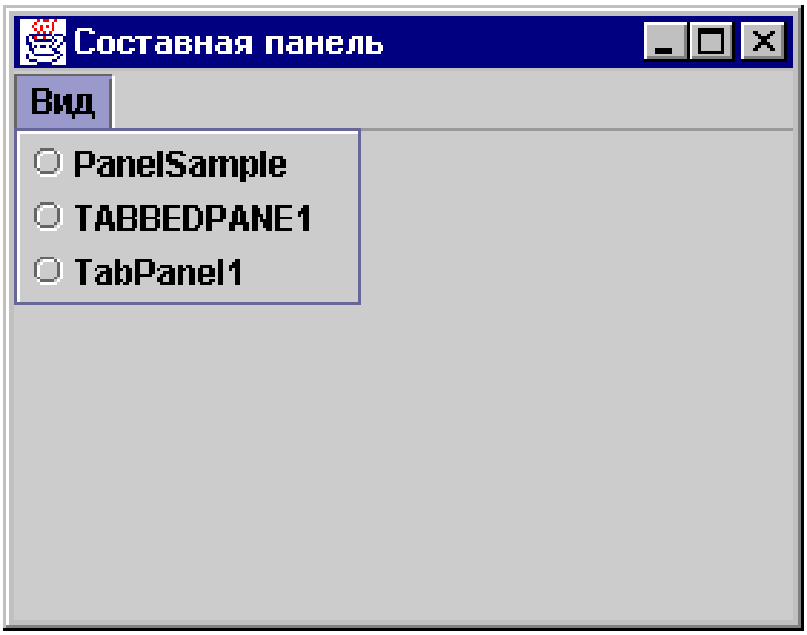
From the menu bar on the GUI Builder **File** window, click the **Insert Deck Pane** tool button  to display a panel builder where you can insert the components for your deck pane. В приведенном ниже примере добавляется три компонента.

Рисунок 1: Создание составной панели с помощью GUI Builder



После создания составной панели нажмите кнопку  инструмента **Предварительный просмотр** для вывода панели на экран. Составная панель будет пустой, пока вы не откроете меню **Вид**.

Рисунок 2: Предварительный просмотр составной панели с помощью GUI Builder



В меню **Вид** составной панели выберите элемент для просмотра. В этом примере можно выбрать элемент PanelSample, TABBEDPANE1, или TablePanel. На приведенных ниже рисунках показано, как эти элементы будут выглядеть при предварительном просмотре.

Рисунок 3: Просмотр элемента PanelSample с помощью GUI Builder

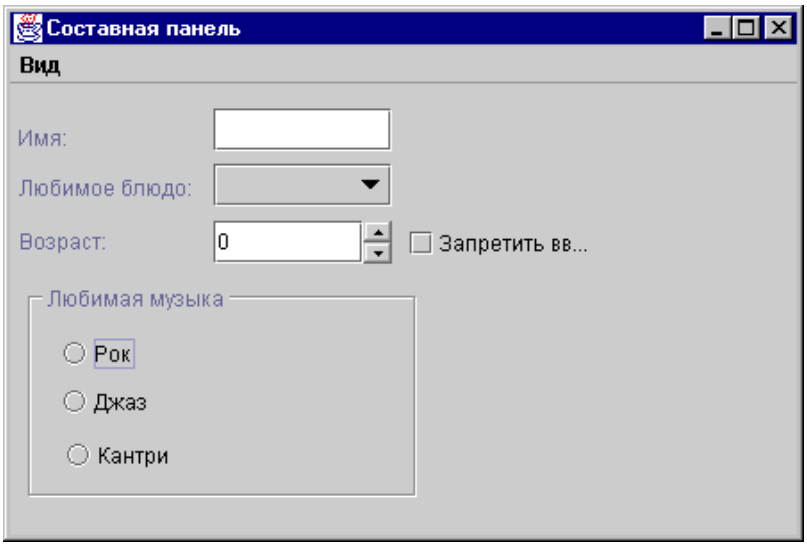


Рисунок 4: Просмотр TABBEDPANE1 с помощью GUI Builder

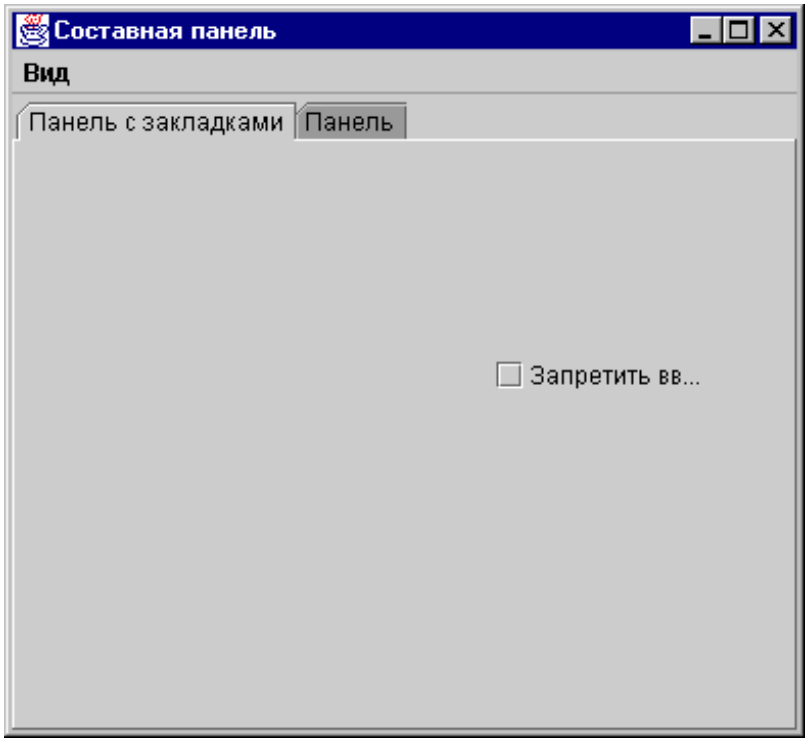
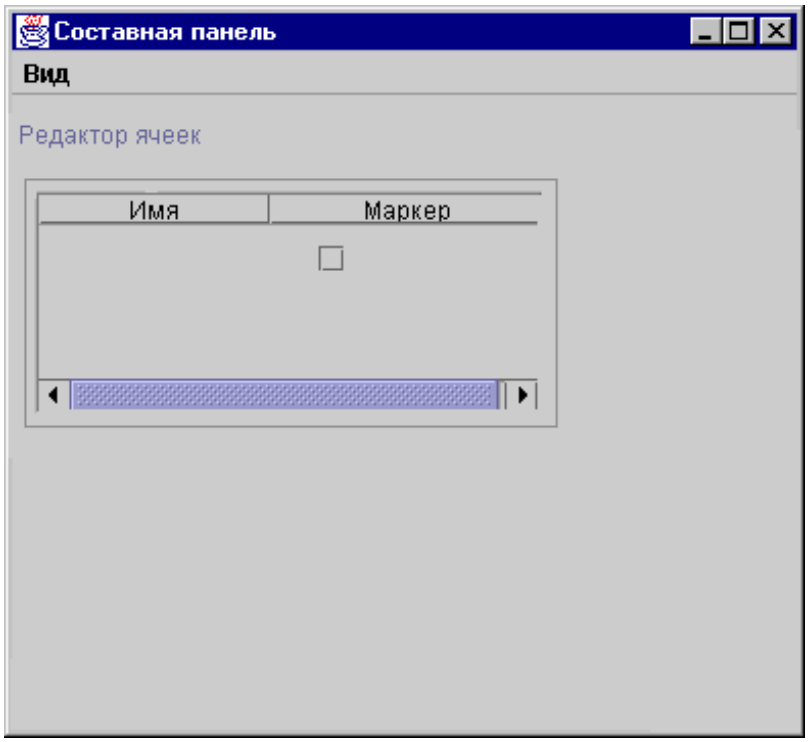


Рисунок 5: Просмотр элемента TablePanel с помощью GUI Builder



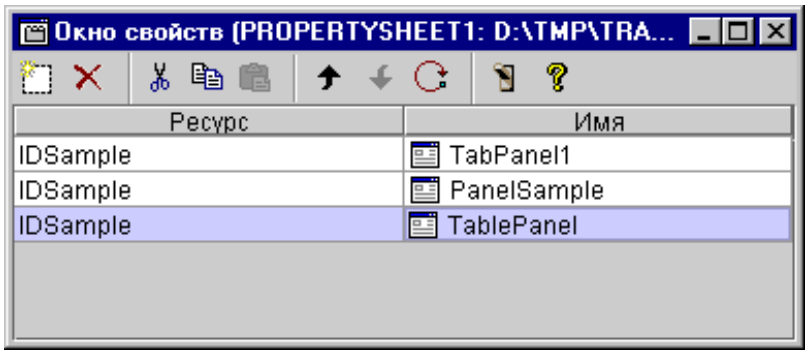
Создание окна свойств с помощью GUI Builder

Follow these steps to create a property sheet using GUI Builder.

Программа GUI Builder позволяет упростить процедуру создания окна свойств. From the menu bar on the main GUI Builder window, select **File --> New File**.

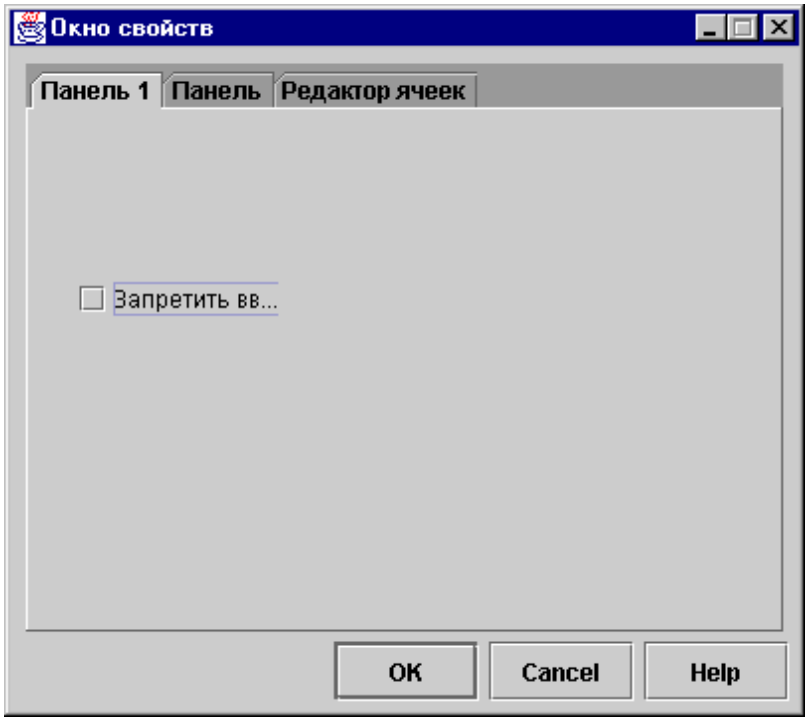
В строке меню окна **Файл** GUI Builder нажмите кнопку Вставить разделенную панель  . Появится Редактор панелей, в котором можно можно добавить компоненты разделенной панели.

Рис. 1: Создание окна свойств с помощью GUI Builder



Для того чтобы просмотреть созданное окно свойств, щелкните на значке  . В данном примере вы можете выбрать один из трех ярлыков.

Рис. 2: Предварительный просмотр окна свойств с помощью GUI Builder



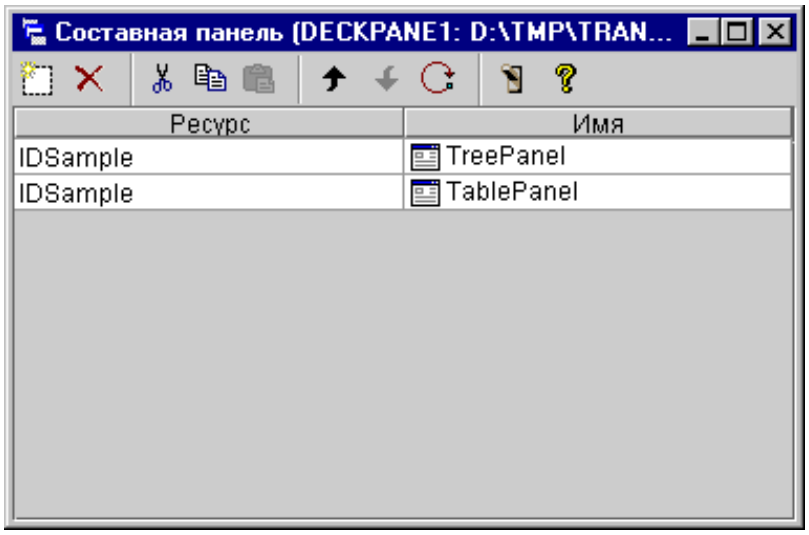
Создание разделенной панели в GUI Builder

Программа GUI Builder упрощает создание разделенных панелей.

From the menu bar on the main GUI Builder window, select **File --> New File**.

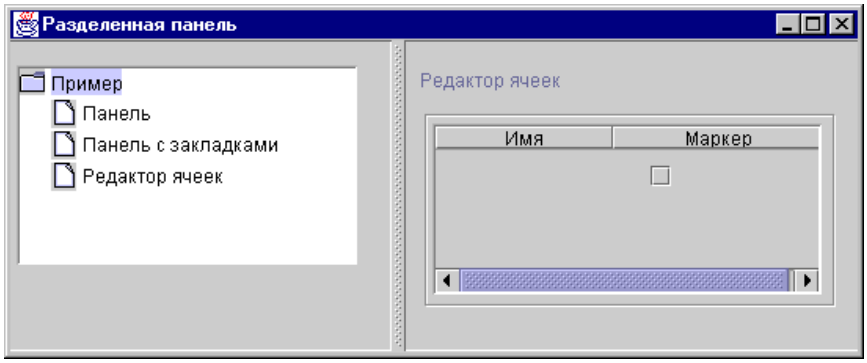
В меню **Файл** GUI Builder щелкните на значке Вставить новую панель  для перехода к окну создания панелей, в котором можно добавлять новые компоненты разделенной панели. В следующем примере добавляются два компонента.

Figure 1: Creating a Split Pane with GUI Builder



After you create the split pane, click the **Preview** tool button  icon to preview it, as shown in Figure 2.

Figure 2: Previewing the Split Pane with GUI Builder



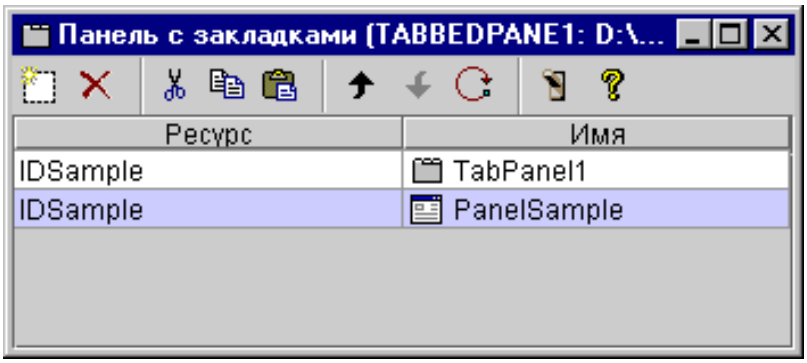
Создание панели со вкладками с помощью GUI Builder

Use these instruction to create a tabbed pane.

В строке меню главного окна GUI Builder выберите **Файл → Создать файл**.

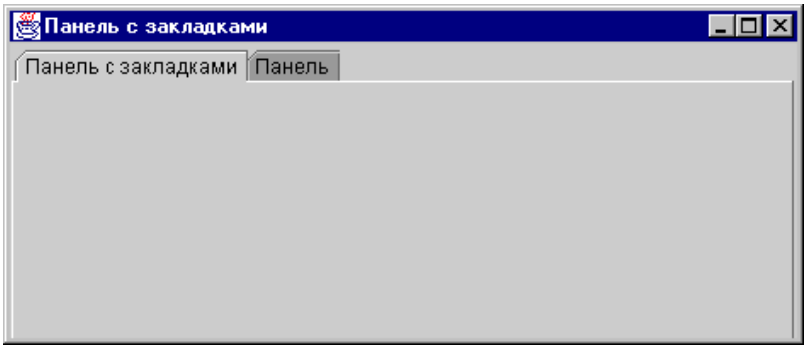
В меню **Файл** GUI Builder щелкните на значке Вставить панель с закладками  для перехода к окну создания панели, в котором в панель можно добавить новые компоненты. В следующем примере добавляются два компонента.

Figure 1: Creating a Tabbed Pane in GUI Builder



After you create the tabbed pane, click the **Preview** tool button to preview it.

Figure 2: Previewing the Tabbed Pane with GUI Builder



Создание мастера с помощью GUI Builder

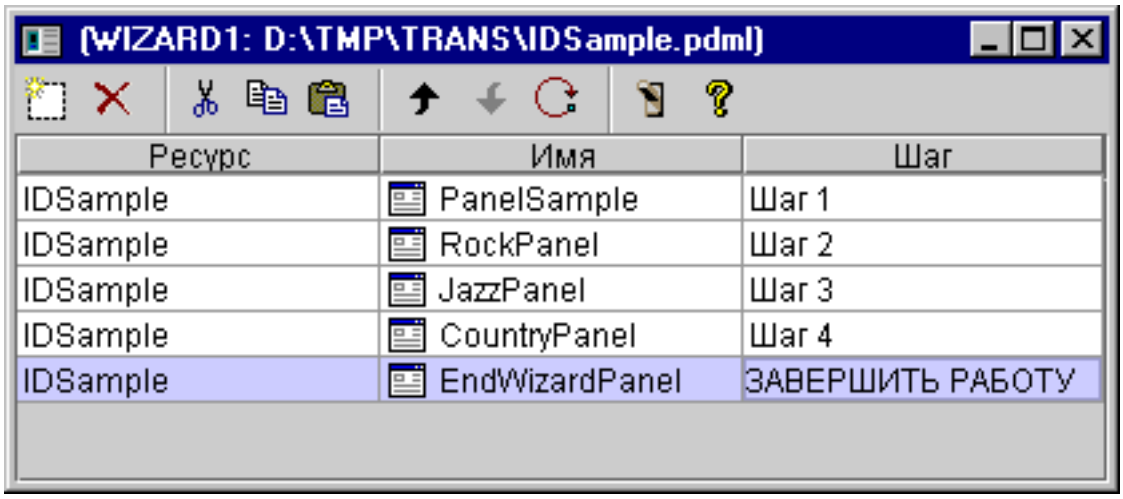
Use these instructions to create a wizard.

Программа GUI Builder упрощает создание программ-мастеров. From the menu bar on the GUI Builder window, select **File --> New File**.

From the menu bar on the GUI Builder **File** window, click the Insert Wizard toolbar button

to display a panel builder where you can add panels to the wizard.

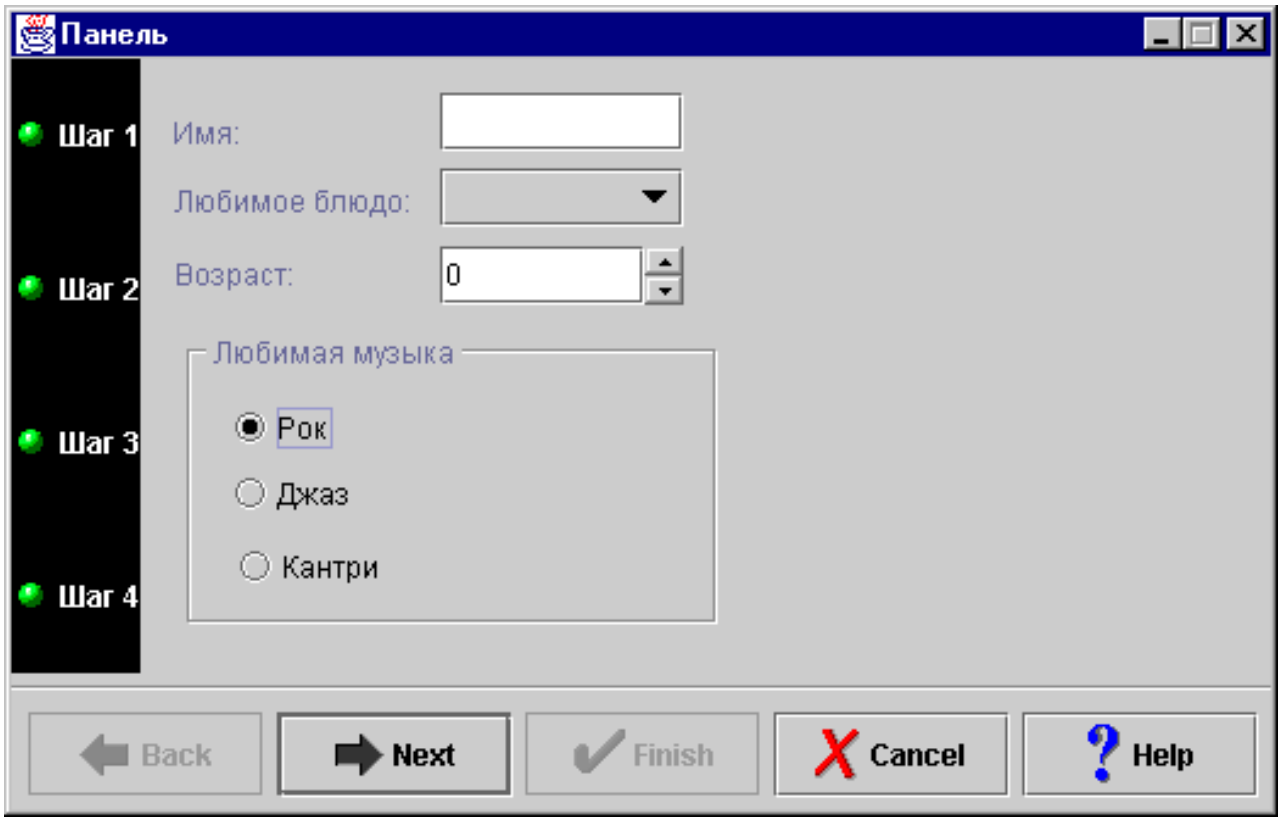
Рисунок 1: Создание мастера с помощью GUI Builder



After you have create the wizard, use the **Preview** tool button

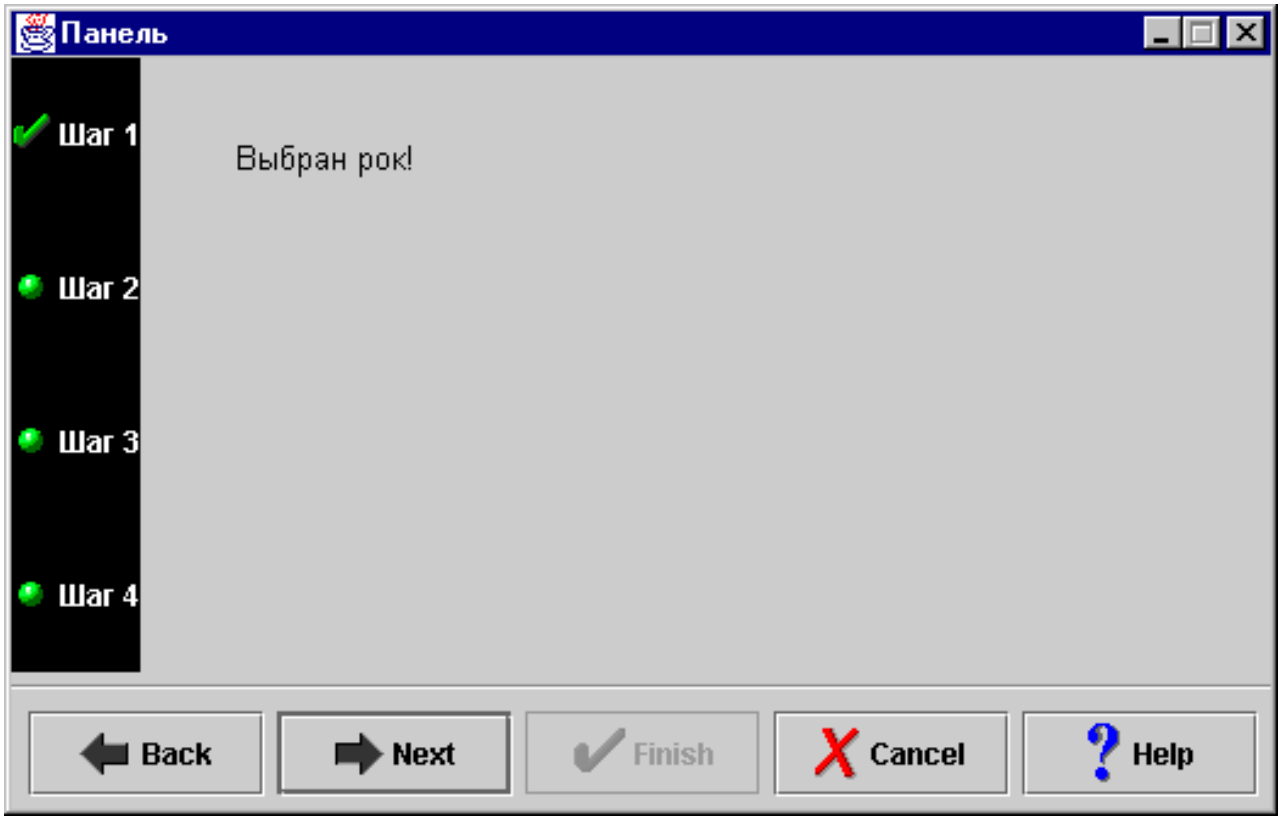
to preview it. На рис. 2 показано окно предварительного просмотра для данного примера.

Рисунок 2: Предварительный просмотр мастера с помощью GUI Builder



На рис. 2 показано второе окно, появляющееся в случае, если пользователь выберет пункт **Рок** и нажмет кнопку **Далее**.

Рисунок 3: Предварительный просмотр второй панели мастера с помощью GUI Builder



Если во втором окне мастера нажать кнопку **Далее**, появится последнее окно, показанное на рис. 4.

Рисунок 4: Предварительный просмотр последней панели мастера с помощью GUI Builder

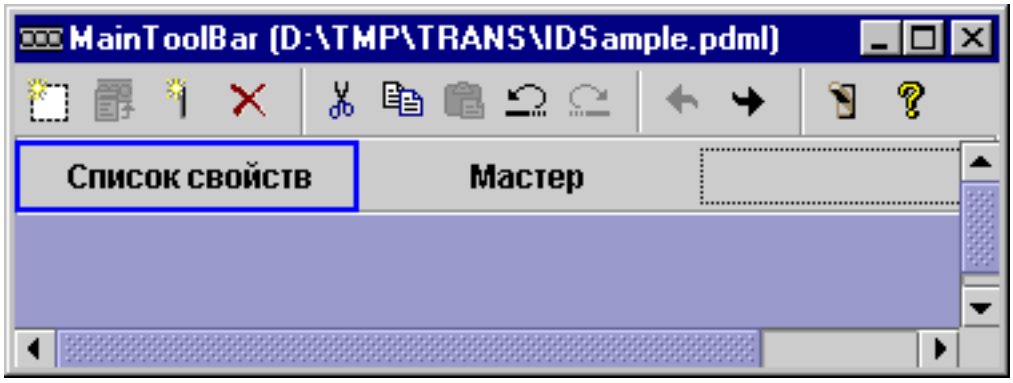
Создание панели инструментов с помощью GUI Builder

Follow these instructions to create a toolbar with GUI Builder.

From the menu bar on the GUI Builder window, select **File → New File**.

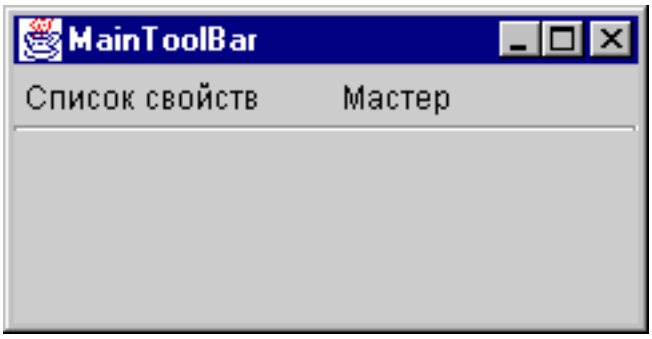
В строке меню окна **Файл** GUI Builder нажмите кнопку **Вставить панель инструментов**. Появится окно создания панели инструментов.

Рисунок 1: Создание панели инструментов с помощью GUI Builder



After you create the toolbar, click the **Preview** tool button  to preview it. В данном случае панель инструментов может вызывать окно свойств или окно мастера.

Рисунок 2: Предварительный просмотр панели инструментов с помощью GUI Builder



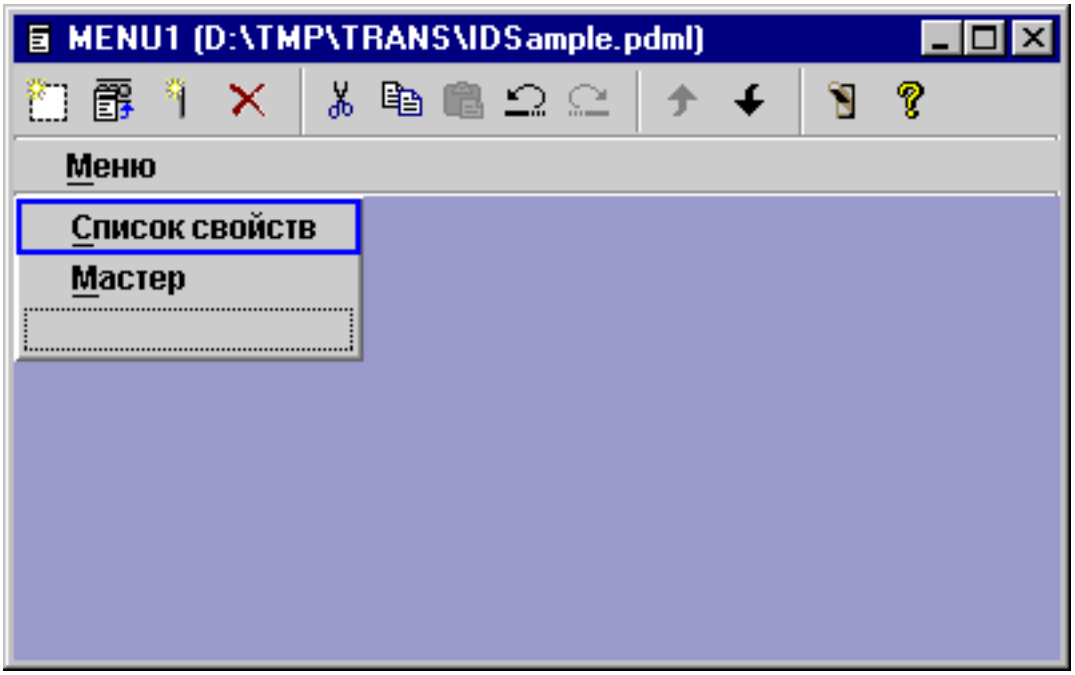
Создание меню с помощью GUI Builder

Use these instructions to create a menu bar.

Программа GUI Builder позволяет легко создавать строки меню. From the menu bar on the GUI Builder window, select **File --> New File**.

В панели инструментов окна **Файл** программы GUI Builder нажмите кнопку **Вставить в меню**. Откроется окно создания панели, с помощью которого вы сможете выбрать компоненты для меню.

Рисунок 1: GUI Builder - Создание меню



После создания меню нажмите кнопку **Предварительный просмотр** инструмента  для вывода меню на экран. В данном примере в только что созданном меню **Запуск** вы можете выбрать **Окно свойств** или **Мастер**. Изображение, появляющееся при выборе этих пунктов меню, приведено на следующих рисунках.

Рисунок 2: GUI Builder - Просмотр Окна свойств меню Запуск

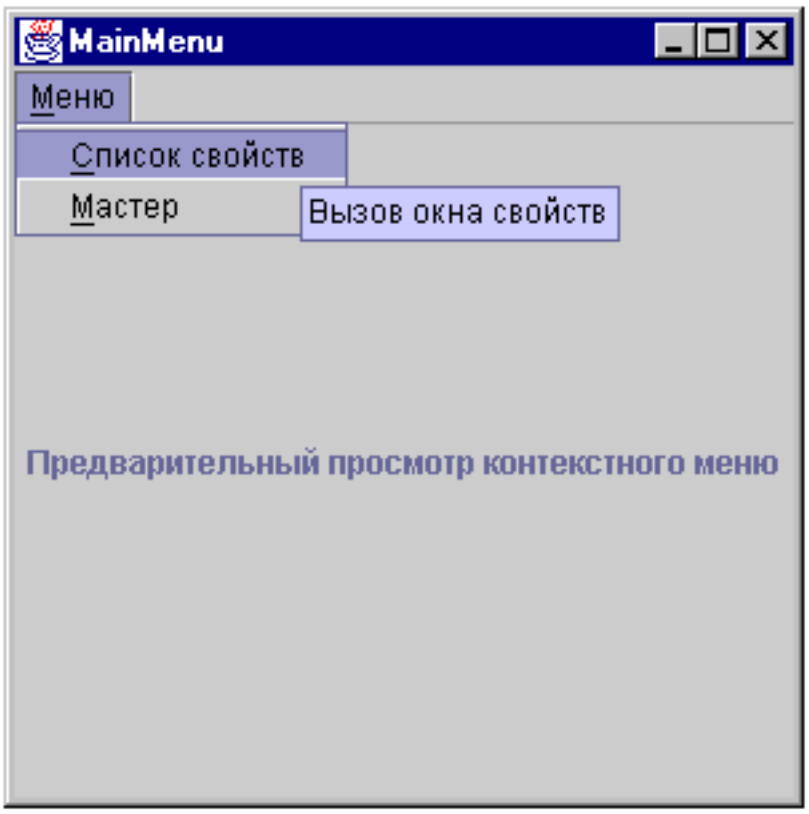
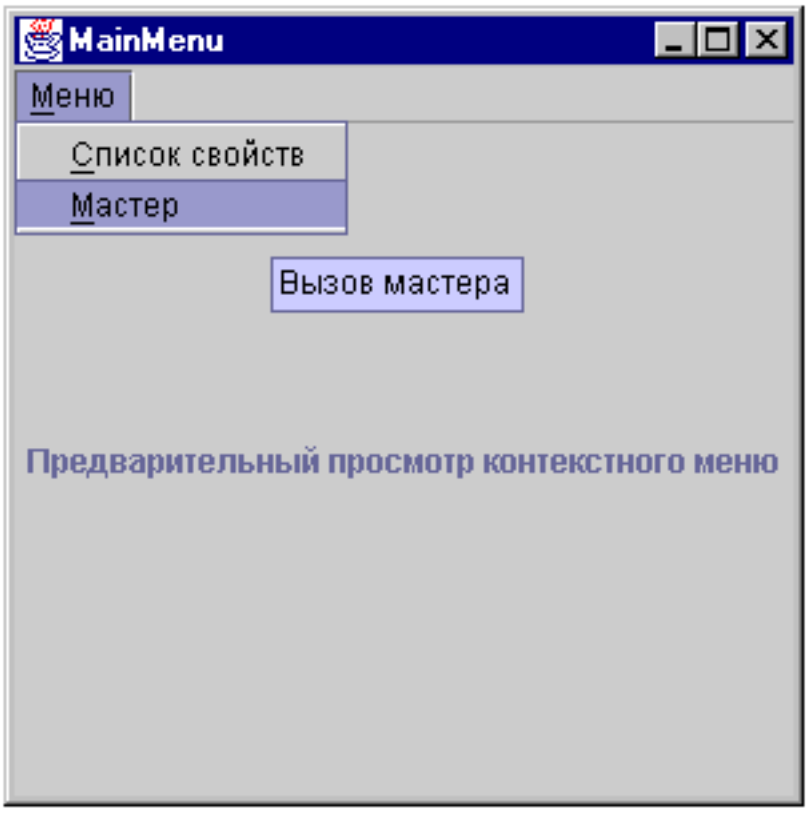


Рисунок 3: GUI Builder - Просмотр Мастера в меню Запуск

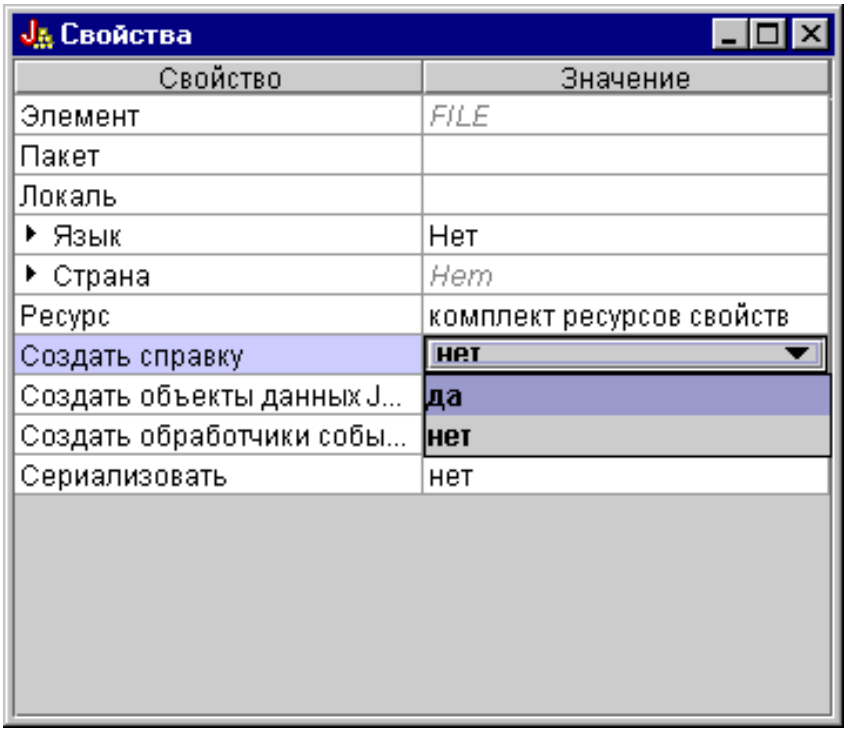


Пример: Создание справки

This topic explains how to create help files with GUI Builder.

GUI Builder позволяет легко создавать файлы справки. На панели свойств файла, с которым вы работаете, включите опцию ″Создать справку″.

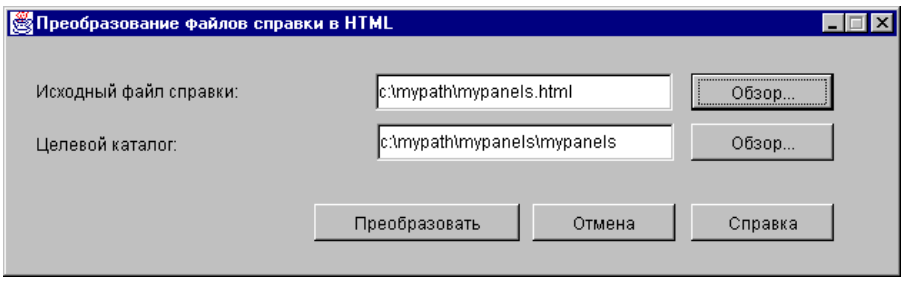
Figure 1: Setting the Generate Help property on the GUI Builder Properties panel



GUI Builder создаст макет документа HTML, называемый справочным документом, который можно отредактировать.

Для использования справки во время работы программы необходимо поместить разделы, определенные внутри файла PDML, в отдельные файлы HTML. При запуске процедуры **Преобразование документа справки в HTML** для разделов формируются индивидуальные файлы HTML, которые помещаются в подкаталог, указанный после документа справки и файла PDML. Именно в этом каталоге среда выполнения программы будет искать файлы справки. В окне диалога **Преобразование документа справки в HTML** вводится вся необходимая информация, после чего запускается программа HelpDocSplitter:

Figure 2: Help Document to HTML Processing dialog



Для запуска программы обработки справочного файла необходимо ввести в командной строке:

```
jre com.ibm.as400.ui.tools.hdoc2htmViewer
```

Перед запуском программы необходимо правильно задать переменную среды CLASSPATH.

Для обработки справочного документа необходимо сначала выбрать документ с именем, совпадающим с именем файла PDML. После этого требуется указать целевой каталог, имя которого должно состоять из имени справочного файла и имени файла PDML. Для запуска процесса обработки выберите ″Обработать″.

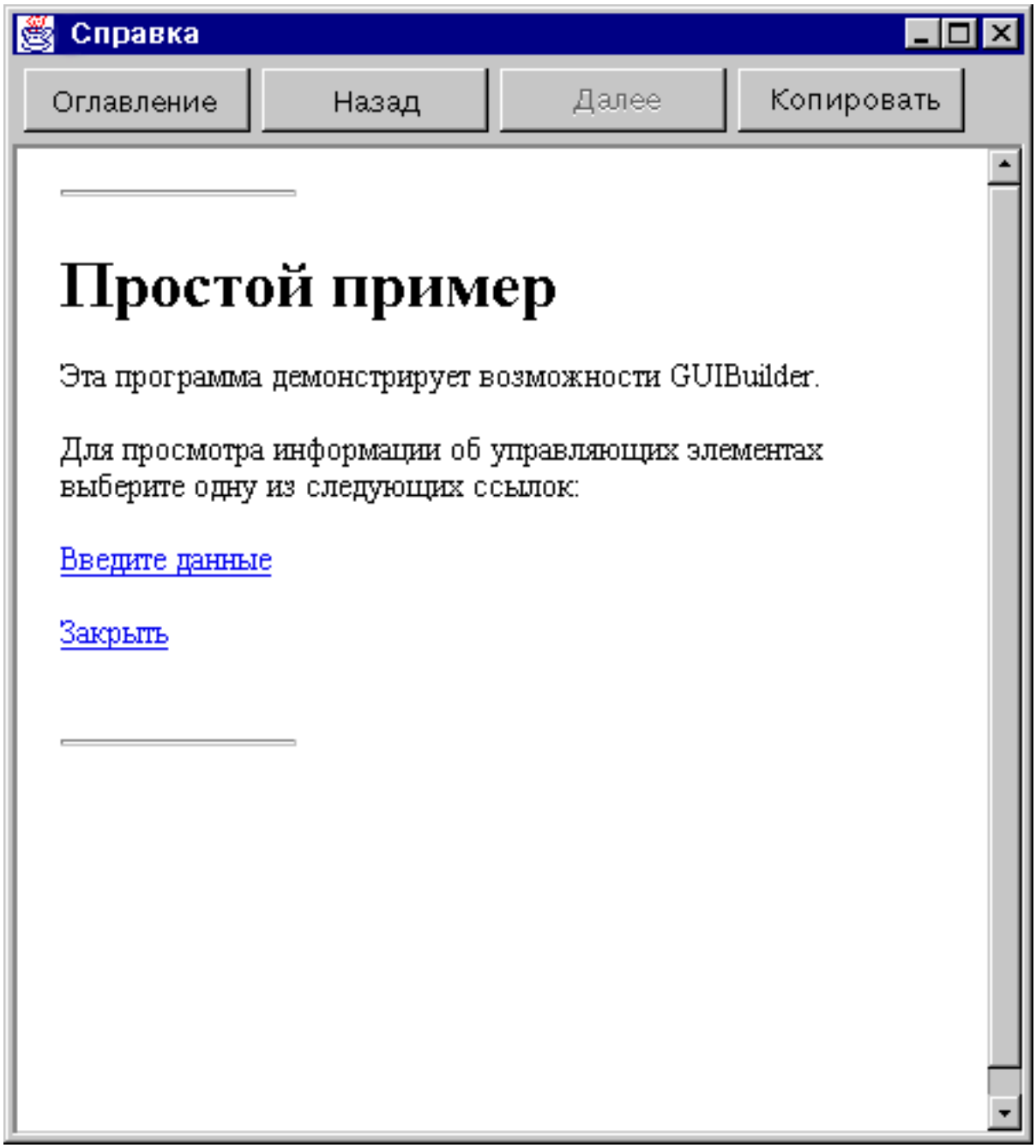
Вы можете разбить исходный файл справки на подразделы, введя в командной строке следующую команду:

```
jre com.ibm.as400.ui.tools.HelpDocSplitter "имя_справочного_документа.htm" [целевой каталог]
```

Эта команда запустит функцию, разбивающую исходный файл на файл HTML по темам. Имя справочного файла передается в одном из аргументов команды. Кроме него, можно указать целевой каталог. По умолчанию создается каталог с именем, совпадающим с именем входного файла, и при обработке файлы помещаются в этот каталог.

Ниже приведен пример файла справки:

Рисунок 3: Пример файла справки GUI Builder



Пример: Применение GUI Builder

Для того чтобы получить полноценное приложение GUI, к примерам, приведенным в данном разделе, нужно добавить необходимые компоненты данных.

На рис. 1 показано первое окно, появляющееся при выполнении данного примера.

При изменении размера панели и управляющих элементов с помощью динамического администратора панели размер текста не меняется.

На этой панели можно выполнить следующие действия:

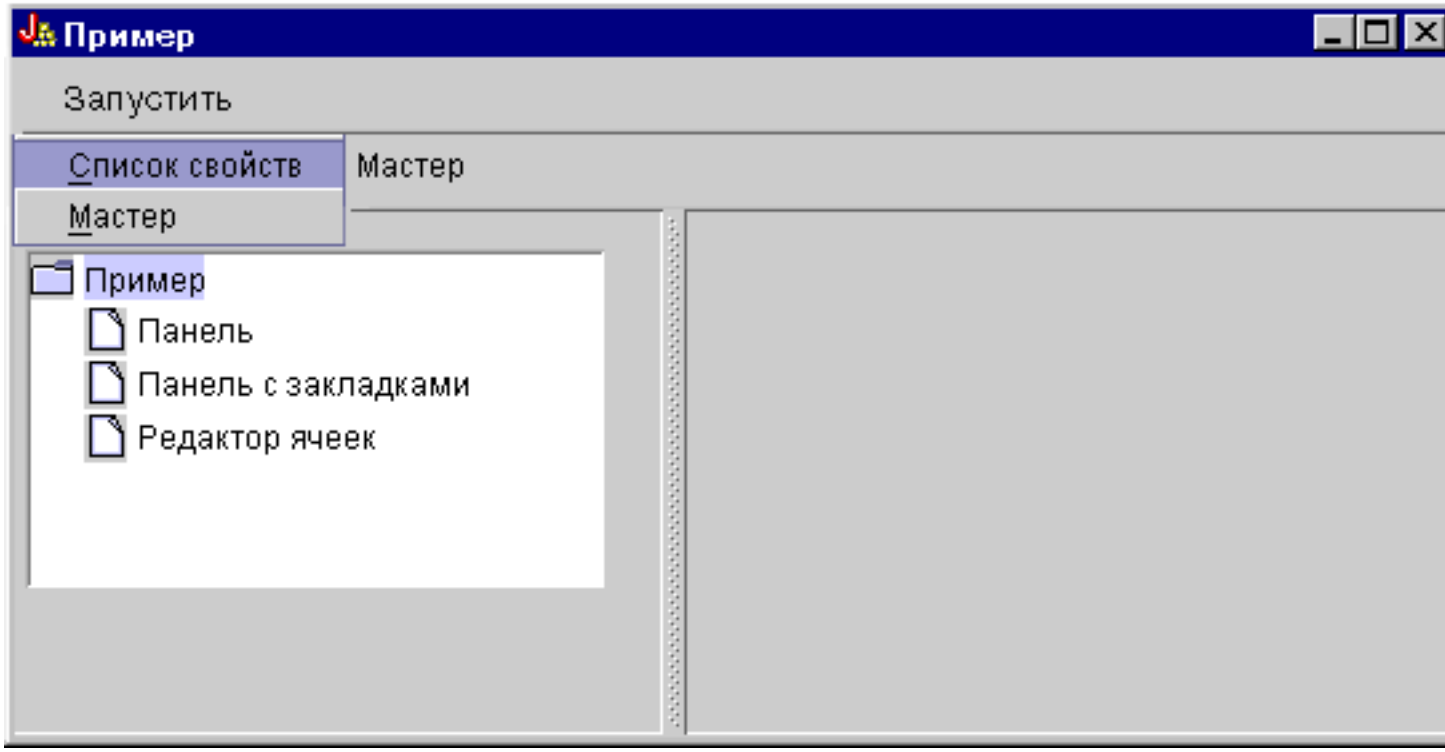
- Открыть окно свойств
- Запустить мастер

- Показать примеры, перечисленные в левой панели

Открыть окно свойств

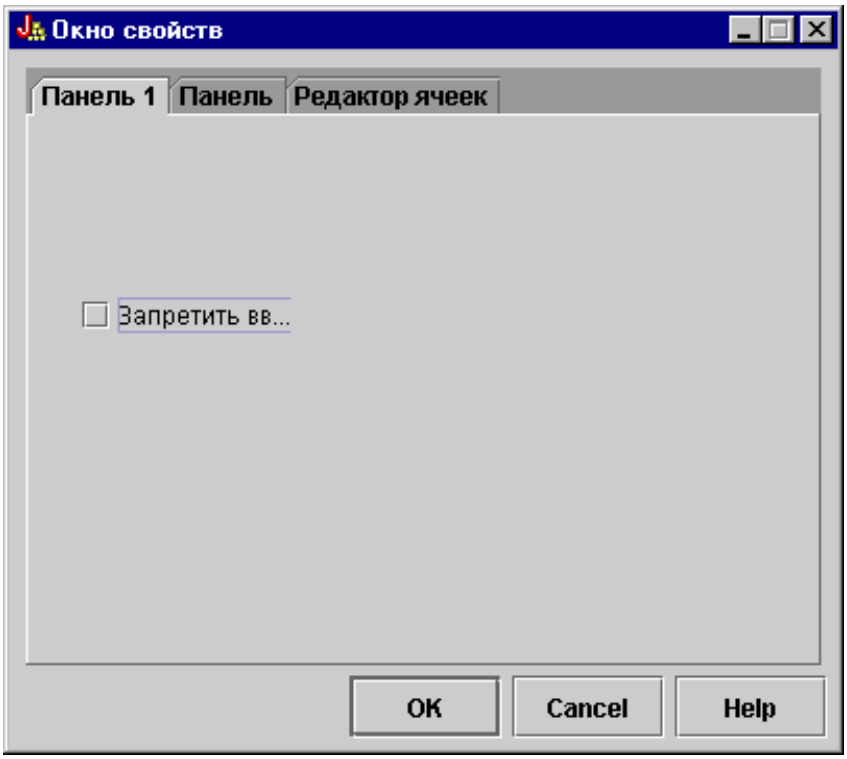
Для создания окна свойств нужно нажать кнопку Окно свойств на панели инструментов или воспользоваться меню **Запустить**. В этом примере демонстрируется связь между пунктами меню и элементами панели инструментов. На рис. 4 показано **окно свойств**, выбранное из меню **Запустить** главного окна GUI Builder.

Рисунок 4: Создание окна свойств с помощью меню Запустить



После выбора пункта **Окно свойств** будет показано окно, изображенное на рис. 5.

Рисунок 5: Пример окна свойств



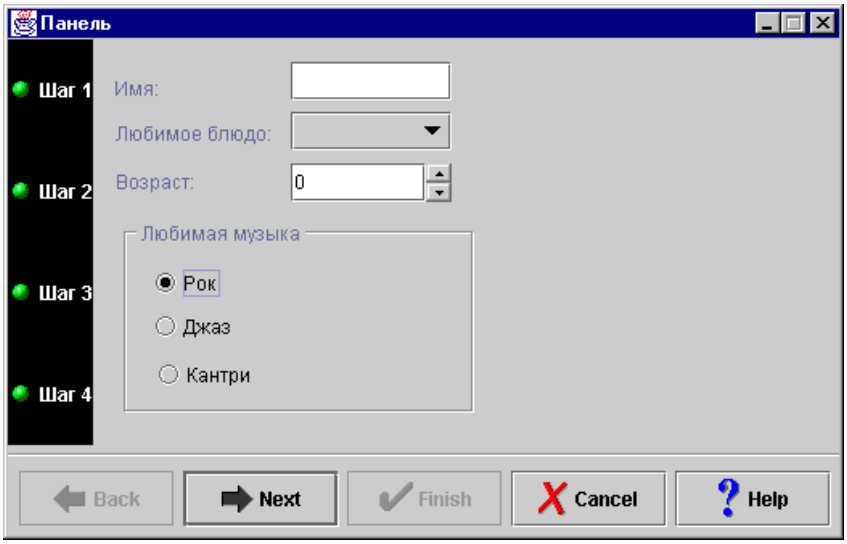
Первоначально в окне свойств открыта первая вкладка.

Запуск мастера

Мастер можно запустить с помощью кнопки на панели инструментов или из меню **Запустить**. В этом примере демонстрируется связь между пунктами меню и элементами панели инструментов.

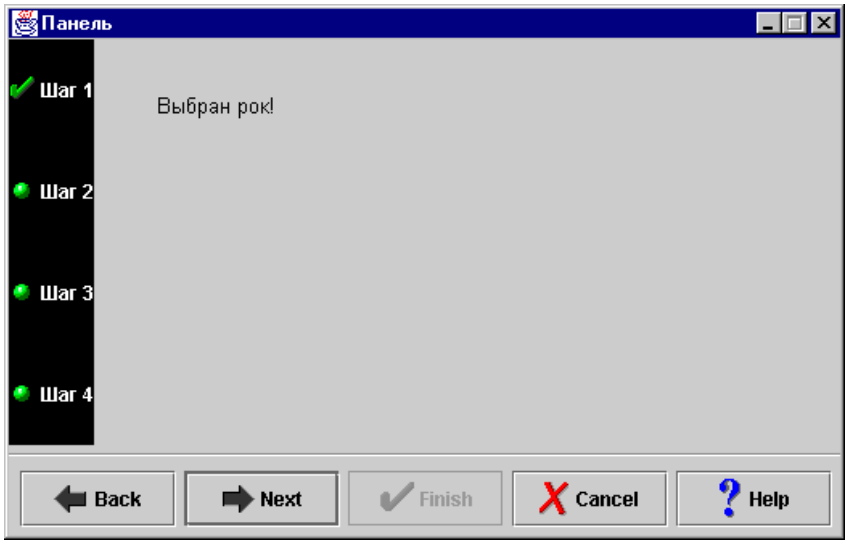
На рис. 9 показаны опции первого окна мастера.

Рисунок 9: Выбор опции Рок в первом окне мастера



Выберите опцию **Рок** в первом окне мастера и нажмите кнопку **Далее** для перехода во второе окно, показанное на рис. 10.

Рисунок 10: второе окно мастера (после выбора опции Рок)



Нажмите кнопку **Далее** во втором окне диалога для перехода в последнее окно, показанное на рис. 11.

Рисунок 11: Последнее окно мастера

В данном примере предусмотрен один цикл. Выберите опцию **Страна** в первом окне мастера (рис. 12), затем нажмите кнопку **Далее** для перехода во второе окно (рис. 13). Если вы нажмете кнопку Далее во втором окне, вновь появится первое окно (рис. 14).

Рисунок 12: Выбор опции Кантри в первом окне мастера

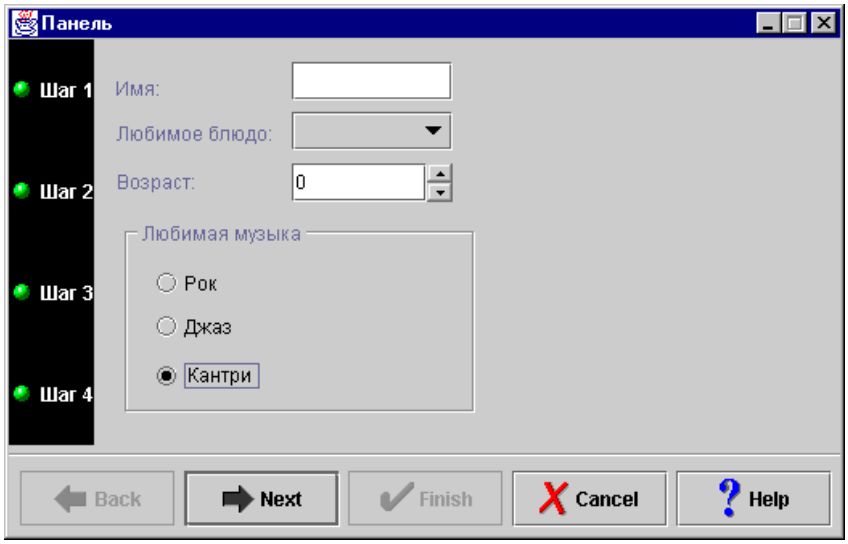


Рисунок 13: второе окно мастера (после выбора опции Кантри)

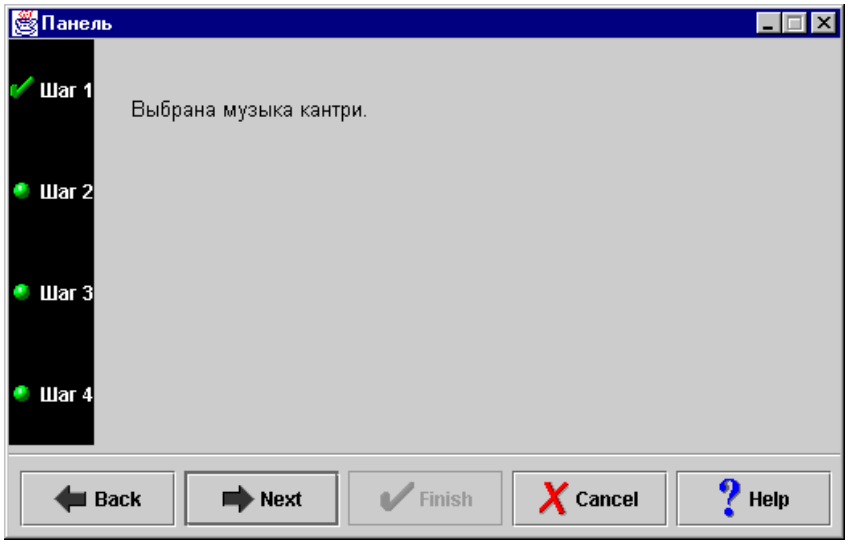
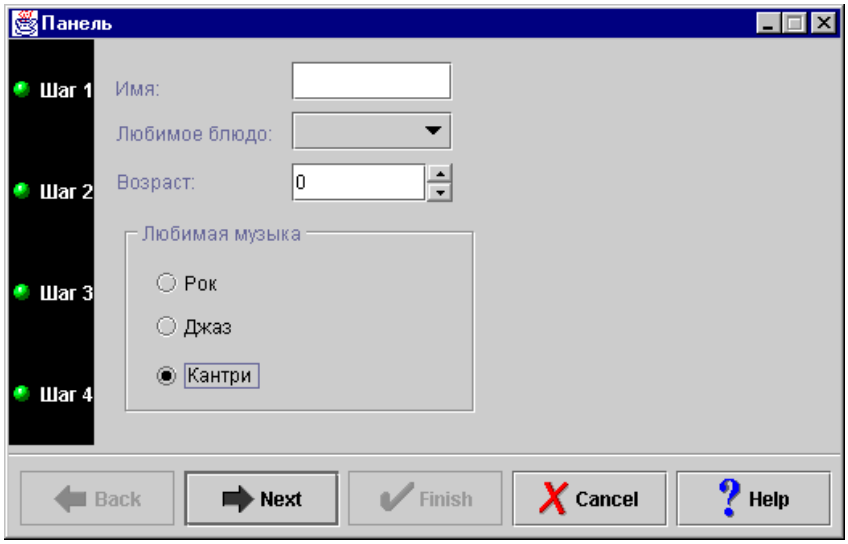


Рисунок 14: Возврат в первое окно мастера

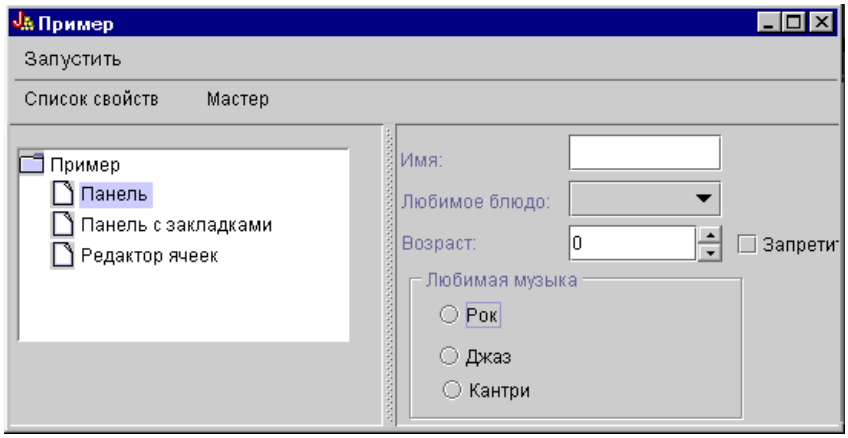


Это означает, что программист запретил выбирать кантри в качестве любимого стиля музыки.

Просмотр примеров

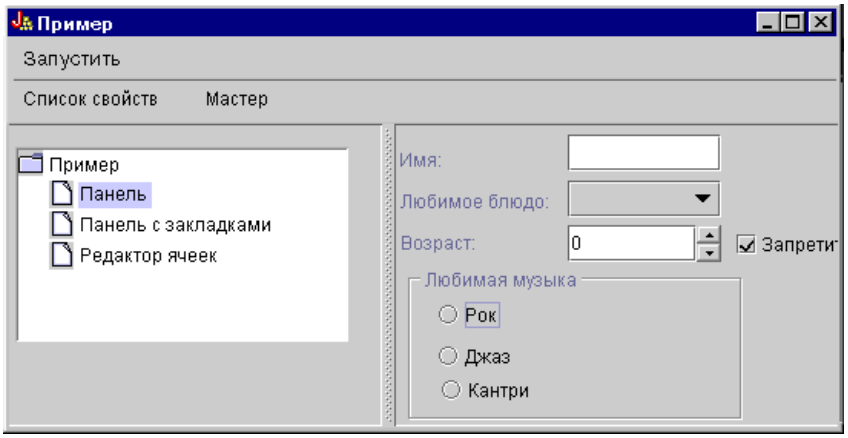
В левой панели главного окна примера можно выбрать и другие функции. На рис. 15 показано окно, которое появится после выбора пункта **Панель** в левой панели окна.

Рисунок 15: выбор опции Панель в левой панели окна



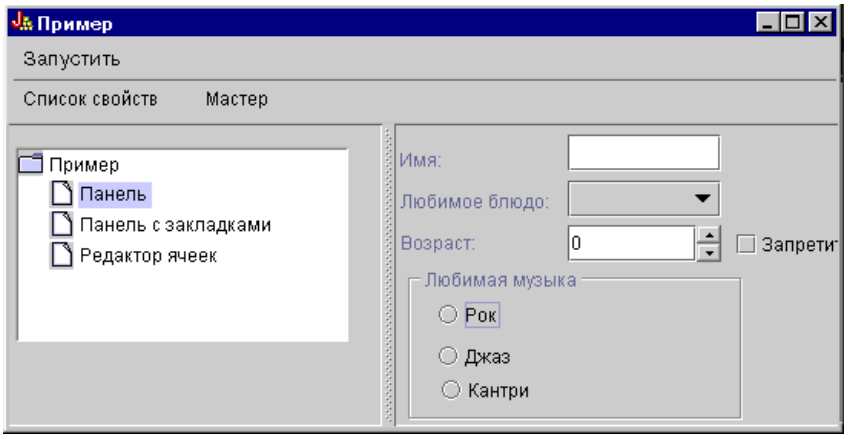
В этом примере предусмотрена опция, отключающая вывод изображений. Если вы выберете опцию **Отключить вывод изображений**, то изображения не будут показаны в окне (см. рис. 16).

Рисунок 16: выбор опции Отключить вывод изображений в правой панели



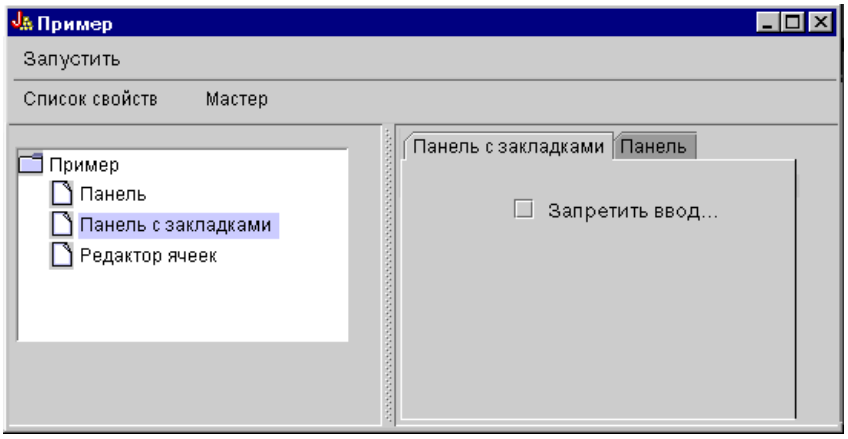
Данный пример также демонстрирует выпадающий список (см. рис. 17).

Рисунок 17: Выбор элемента в списке Любимое блюдо в правой панели



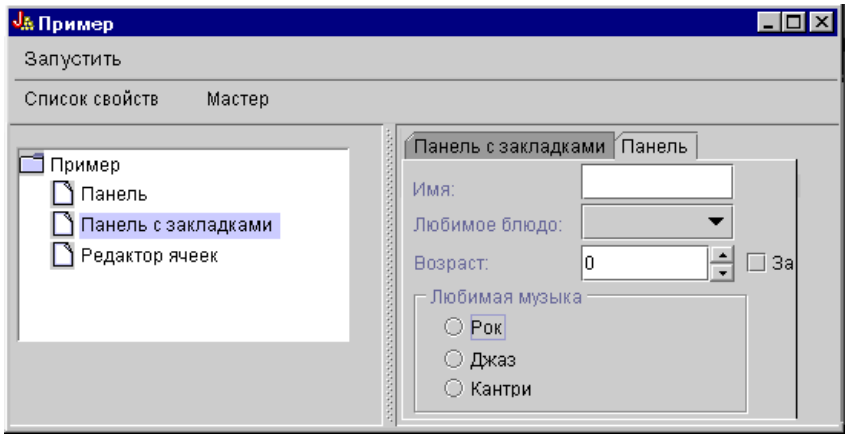
На рис. 18 показано окно, появляющееся после выбора опции **Панель с закладками** в левой панели примера.

Рисунок 18: выбор опции Панель с закладками в левой панели



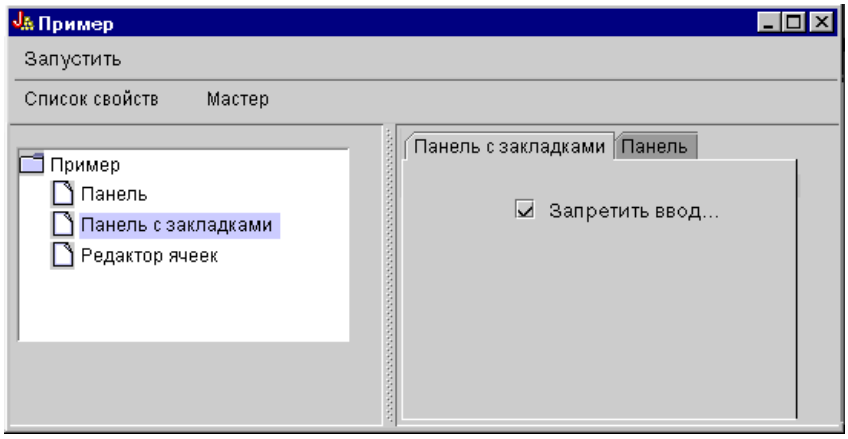
На рис. 19 показано окно, появляющееся после выбора вкладки **Пример панели** в правой панели.

Рисунок 19: вкладка Пример панели в правой панели



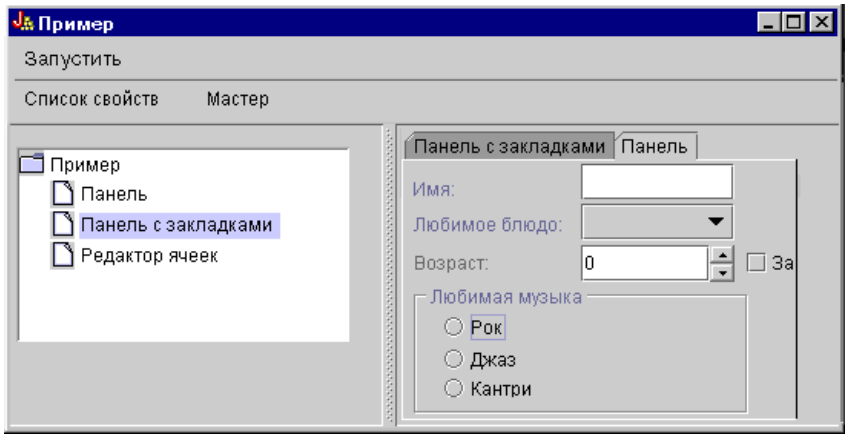
Вновь выберите **Вкладку 1** (в правой панели), затем щелкните в поле **Запретить ввод возраста на вкладке 2**.

Рисунок 20: Выбор опции Запретить ввод возраста на вкладке 2 в правой панели



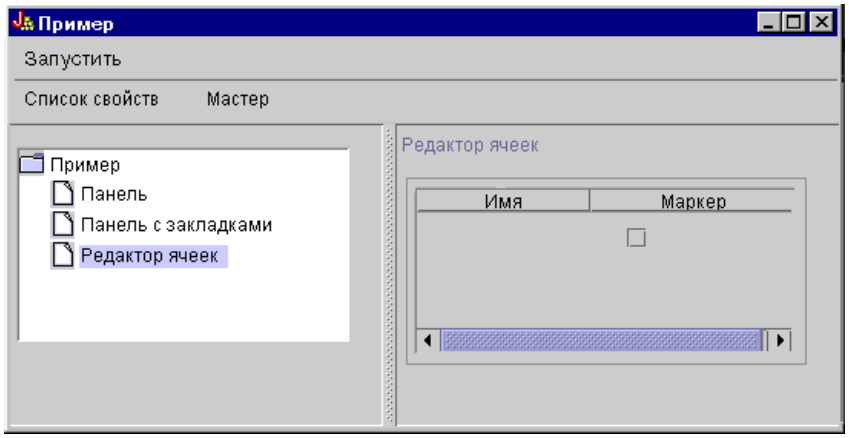
После выбора опции **Запретить ввод возраста на вкладке 2** поле **Возраст** на вкладке **Пример панели** станет недоступным (см. рис. 21).

Рисунок 21: результат отключения поля возраста



Выберите в левой области окна опцию **Таблица**. Появится панель с таблицей, в которой применяется пользовательский способ ввода и пользовательский редактор ячеек (см. рис. 22).

Рисунок 22: выбор опции Таблица в левой панели



Примеры применения классов HTML

The following examples show you some of the ways that you can use the IBM Toolbox for Java HTML classes.

- Example: Using the BidiOrdering class
- Пример: Создание объектов HTMLAlign
- Примеры применения класса HTMLDocument:
  - Пример: Создание данных HTML с помощью класса HTMLDocument
  - Пример: Создание данных XSL FO с помощью класса HTMLDocument
- Пример: Применение классов форм HTML
- Примеры применения классов элемента ввода:
  - Пример: Создание объекта ButtonFormInput
  - Пример: Создание объекта FileFormInput
  - Пример: Создание объекта HiddenFormInput
  - Пример: Создание объекта ImageFormInput
  - Пример: Создание объекта ResetFormInput
  - Пример: Создание объекта SubmitFormInput
  - Пример: Создание объекта TextFormInput
  - Пример: Создание объекта PasswordFormInput
  - Пример: Создание объекта RadioFormInput
  - Пример: Создание объекта CheckboxFormInput
- Пример: Создание объектов HTMLHeading
- Пример: Применение класса HTMLHyperlink
- Пример: Применение класса HTMLImage
- Примеры HTMLList
  - Пример: Создание упорядоченных списков
  - Пример: Создание неупорядоченных списков
  - Пример: Создание вложенных списков
- Пример: Создание тегов HTMLMeta
- Пример: Создание тегов HTMLParameter
- Пример: Создание тегов HTMLServlet
- Пример: Применение класса HTMLText
- Примеры HTMLTree

  - Пример: Применение класса HTMLTree
  - Пример: Создание просматриваемого дерева интегрированной файловой системы
- Классы макетов форм:
  - Пример: Применение класса GridLayoutFormPanel
  - Пример: Применение класса LineLayoutFormPanel
- Пример: Применение класса TextAreaFormElement
- Пример: Применение класса LabelFormOutput
- Пример: Применение класса SelectFormElement
- Пример: Применение класса SelectOption
- Пример: Применение класса RadioFormInputGroup
- Пример: Применение класса RadioFormInput
- Пример: Применение классов HTMLTable
  - Пример: Применение класса HTMLTableCell
  - Пример: Применение класса HTMLTableRow
  - Пример: Применение класса HTMLTableHeader
  - Пример: Применение класса HTMLTableCaption

Кроме того, классы HTML могут применяться совместно с классами сервлета, как показано в данном примере.

The following disclaimer applies to all of the IBM Toolbox for Java examples:

> **Отказ от гарантий на предоставляемый код**
>
> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.
>
> Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Применение классов форм HTML

The following IBM Toolbox for Java example shows you how to use the HTML form classes.

а также пример вывода, создаваемого этим кодом. Классы HTML, использованные в методе ″showHTML″, выделены **полужирным шрифтом**.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
R ////////////////////////////////////////////////////////////////////////////
R //
R // This source is an example of using the IBM Toolbox for Java HTML
R // AS/400 Toolbox для Java для создания форм HTML.
R //
R ////////////////////////////////////////////////////////////////////////////
R
R package customer;
R
R import java.io.*;
R import java.awt.Color;
R
```

```
R import javax.servlet.*;
R import javax.servlet.http.*;
R
R import com.ibm.as400.access.*;
R import com.ibm.as400.util.html.*;
R
R
R public class HTMLExample extends HttpServlet
R {
R   // Определение того, находится ли пользователь в списке регистрантов.
R   private static boolean found = false;
R
R   // Файл для хранения регистрационных данных
R   String regPath = "c:\\registration.txt";
R
R
R   public void init(ServletConfig config)
R   {
R
R     try
R     {
R        super.init(config);
R
R     }
R     catch(Exception e)
R     {
R        e.printStackTrace();
R     }
R
R   }
R
R   /**
R    *  Обработка запроса GET.
R    *  В параметре req - запрос.
R    *  В параметре res - ответ.
R    **/
R    public void doGet (HttpServletRequest req, HttpServletResponse res)
R       throws ServletException, IOException
R    {
R       res.setContentType("text/html");
R       ServletOutputStream out = res.getOutputStream();
R
R       // Получение исходного текста страницы из класса HTML
R       out.println(showHTML());
R       out.close();
R    }
R
R
R     public void doPost (HttpServletRequest req, HttpServletResponse res)
R        throws ServletException, IOException
R     {
R         String nameStr   = req.getParameter("name");
R         String emailStr = req.getParameter("email");
R         String errorText= "";
R
R         // Поток передачи данных сервлету
R         ServletOutputStream out = res.getOutputStream();
R
R         res.setContentType("text/html");
R
R         // Проверка имени и электронного адреса клиента
R         if (nameStr.length() == 0)
R            errorText += "Не указано имя пользователя. ";
R         if (emailStr.length() == 0)
R            errorText += "Не указан электронный адрес. " ;
R
R         // Если имя и электронный адрес указаны, то - продолжение
```

```
R          if (errorText.length() == 0)
R          {
R
R             try
R             {
R                // Создание файла registration.txt
R                FileWriter f = new FileWriter(regPath, true);
R                BufferedWriter output = new BufferedWriter(f);
R
R                // Буферизованное чтение для поиска в файле
R                BufferedReader in = new BufferedReader(new FileReader(regPath));
R
R                String line = in.readLine();
R
R                // Сброс флага found
R                found = false;
R
R                // Проверка, не зарегистрирован ли уже клиент
R                // с тем же именем или электронным адресом
R                while (!found)
R                {
R                   // если файл пуст или просмотрен полностью
R                   if (line == null)
R                      break;
R
R                   // если клиент уже зарегистрирован
R                   if ((line.equals("Имя клиента: " + nameStr)) ||
R                      (line.equals("Электронный адрес: " + emailStr)))
R {
R                      // Вывод сообщения с информацией о том,
R                      // что он уже зарегистрирован
R                      out.println ("<HTML> " +
R                                   "<TITLE> Toolbox Registration</TITLE> " +
R                                   "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> " +
R                                   "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> " );
R                      out.println ("<P><HR>" +
R                                   "<P>" + nameStr +
R                                   "</B>, you have already registered using that " +
R                                   "<B>Name</B> or <B>E-mail address</B>." +
R                                   "<P> Thank You!...<P><HR>");
R
R                      // Создание и вывод объекта HTMLHyperlink
R                      out.println ("<UL><LI>" +
R                                   new HTMLHyperlink("./customer.HTMLExample",
R                                                     "Вернуться к форме регистрации") +
R                                   "</UL></BODY></HTML>");
R                      found = true;
R                      break;
R
R                   }
R                   else    // чтение следующей строки
R                      line = in.readLine();
R
R                }
R
R                // Объект String для хранения данных, полученных из формы HTML
R                String data;
R
R                // Если имени или электронного адреса пользователя нет
R                // в списке, продолжить выполнение операции.
R                if (!found)
R                {
R                   //-------------------------------------------------------------
R                   // Добавление данных о новом клиенте в файл
R                   output.newLine();
R                   output.write("Имя клиента: " + nameStr);
R                   output.newLine();
```

```
R                   output.write("Электронный адрес: " + emailStr);
R                   output.newLine();
R                   //-------------------------------------------------------------
R
R
R                   //-------------------------------------------------------------
R                   // Получение значения переключателя "USE" формы
R                   data = req.getParameter("use");
R                   if(data != null)
R                   {
R                      output.write("Работает с AS/400 Toolbox для Java: " + data);
R                      output.newLine();
R                   }
R                   //-------------------------------------------------------------
R
R
R                   //-------------------------------------------------------------
R                   // Получение значения переключателя "Требуется дополнительная информация"
R                   data = req.getParameter("contact");
R                   if (data != null)
R                   {
R                      output.write("Требуется дополнительная информация: " + data);
R                      output.newLine();
R                   }
R                   //-------------------------------------------------------------
R
R
R                   //-------------------------------------------------------------
R                   // Получение значения поля "Версия AS400" формы
R                   data = req.getParameter("version");
R                   if (data != null)
R                   {
R                      if (data.equals("multiple versions"))
R                      {
R                         data = req.getParameter("MultiList");
R                         output.write("Применяемые версии: " + data);
R                      }
R                      else
R                         output.write("Версия AS400: " + data);
R
R                      output.newLine();
R                   }
R                   //-------------------------------------------------------------
R
R
R                   //-------------------------------------------------------------
R                   // Получение значения поля "Область применения Java" формы
R                   data = req.getParameter("interest");
R                   if (data != null)
R                   {
R                      output.write("Область текущего или будущего применения Java: " + data);
R                      output.newLine();
R                   }
R                   //-------------------------------------------------------------
R
R
R                   //-------------------------------------------------------------
R                   // Получение значения поля "Платформы"
R                   data = req.getParameter("platform");
R                   if (data != null)
R                   {
R                      output.write("Платформы: " + data);
R                      output.newLine();
R                      if (data.indexOf("Other") >= 0)
R                      {
R                         output.write("Другие платформы: " + req.getParameter("OtherPlatforms"));
R                         output.newLine();
```

```
R                   }
R                }
R                //-------------------------------------------------------------
R
R
R                //-------------------------------------------------------------
R                //Getting "Number of System i servers" from form
R                data = req.getParameter("list1");
R                if (data != null)
R                {
R                   output.write("Number of System i servers: " + data);
R                   output.newLine();
R                }
R                //-------------------------------------------------------------
R
R
R                //-------------------------------------------------------------
R                // Получение значения поля "Комментарии" формы
R                data = req.getParameter("comments");
R                if (data != null && data.length() > 0)
R                {
R                   output.write("Комментарии: " + data);
R                   output.newLine();
R                }
R                //-------------------------------------------------------------
R
R
R                //-------------------------------------------------------------
R                // Получение значение поля "Вложение"
R                data = req.getParameter("myAttachment");
R                if (data != null && data.length() > 0)
R                {
R                   output.write("Вложение: " + data);
R                   output.newLine();
R                }
R                //-------------------------------------------------------------
R
R
R                //-------------------------------------------------------------
R                // Получение значения скрытого поля "Copyright"
R                data = req.getParameter("copyright");
R                if (data != null)
R                {
R                   output.write(data);
R                   output.newLine();
R                }
R                //-------------------------------------------------------------
R
R
R                output.flush();
R                output.close();
R
R                // Выдача сообщения "Спасибо!"
R                out.println("<HTML>");
R                out.println("<TITLE>Thank You!</TITLE>");
R                out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
R                out.println("<BODY BGCOLOR=\"blanchedalmond\">");
R                out.println("<HR><P>Thank You for Registering, <B>" + nameStr + "</B>!<P><HR>");
R
R                // Создание и вывод объекта HTMLHyperlink
R                out.println("<UL><LI>" +
R                            new HTMLHyperlink("./customer.HTMLExample", "Вернуться к форме регистрации"));
R                out.println("</UL></BODY></HTML>");
R
R             }
R
R          }
```

```
R            catch (Exception e)
R            {
R               // Вывод сообщение об ошибке в браузере
R               out.println("<HTML>");
R               out.println("<TITLE>ERROR!</TITLE>");
R               out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
R               out.println("<BODY BGCOLOR=\"blanchedalmond\">");
R               out.println("<BR><B>Error Message:</B><P>");
R               out.println(e + "<P>");
R
R               // Создание и вывод объекта HTMLHyperlink
R               out.println("<UL><LI>" +
R                           new HTMLHyperlink("./customer.HTMLExample", "Вернуться к форме регистрации"));
R               out.println("</UL></BODY></HTML>");
R
R               e.printStackTrace();
R            }
R         }
R         else
R         {
R            // Вывод сообщения о том, что имя пользователя и электронный
R            // адрес не указаны, с предложением повторить ввод
R            out.println ("<HTML> " +
R                         "<TITLE>Invalid Registration Form</TITLE> " +
R                         "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> " +
R                         "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> " );
R
R            out.println ("<HR><B>ERROR</B> in customer data - <P><B>" +
R                         errorText +
R                         "</B><P>Please Try Again... <HR>");
R
R
R            // Создание и вывод объекта HTMLHyperlink
R            out.println("<UL><LI>" +
R                        new HTMLHyperlink("./customer.HTMLExample", "Вернуться к форме регистрации"));
R                        "</UL></BODY></HTML>");
R         }
R          // Закрытие потока
R          out.close();
R
R
R      }
R
R
R      public void destroy(ServletConfig config)
R      {
R         // никаких действий выполнять не нужно
R      }
R
R
R      public String getServletInfo()
R      {
R          return "Регистрация продукта";
R      }
R
R      private String showHTML()
R      {
R        // Буфер для хранения Web-страницы
R        StringBuffer page = new StringBuffer();
R
R        // Создание объекта формы HTML.
R        HTMLForm form = new HTMLForm("/servlet/customer.HTMLExample");;
R        HTMLText txt;
R
R        // Создание заголовка Web-страницы и копирование его в буфер
R        page.append("<HTML>\n");
R        page.append("<TITLE> Welcome!!</TITLE>\n");
```

```
R       page.append("<HEAD><SCRIPT LANGUAGE=\"JavaScript\">
R                   function test(){alert(\"Это пример сценария, исполняемого с помощью
R                   ButtonFormInput.\")}</SCRIPT></HEAD>");
R       page.append("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\">\n");
R       page.append("<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"><BR>\n");
R
R       try
R       {
R          //-----------------------------------------------------------------
R          // Создание заголовка страницы с помощью класса HTMLText
R           txt = new HTMLText("Регистрация продукта");
R          txt.setSize(5);
R           txt.setBold(true);
R          txt.setColor(new Color(199, 21, 133));
R          txt.setAlignment(HTMLConstants.CENTER);
R
R          // Вывод текста HTML в буфер
R          page.append(txt.getTag(true) + "<HR><BR>\n");
R          //-----------------------------------------------------------------
R
R          //-----------------------------------------------------------------
R          // Создание разметки строк для имени и электронного адреса
R          LineLayoutFormPanel line = new LineLayoutFormPanel();
R           txt = new HTMLText("Введите свое имя и электронный адрес:");
R          txt.setSize(4);
R          line.addElement(txt);
R
R          // Вывод формы в буфер
R          page.append(line.toString());
R          page.append("<BR>");
R          //-----------------------------------------------------------------
R
R          //-----------------------------------------------------------------
R          // Выбор метода для формы
R          form.setMethod(HTMLForm.METHOD_POST);
R          //-----------------------------------------------------------------
R
R          //-----------------------------------------------------------------
R          // Создание поля для ввода имени.
R          TextFormInput user = new TextFormInput("name");
R          user.setSize(25);
R          user.setMaxLength(40);
R
R          // Создание поля для ввода электронного адреса.
R          TextFormInput email = new TextFormInput("email");
R          email.setSize(30);
R          email.setMaxLength(40);
R
R          // Создание объекта ImageFormInput
R          ImageFormInput img =
R              new ImageFormInput("Отправить форму", "..\\images\\myPiimages/c.gif");
R img.setAlignment(HTMLConstants.RIGHT);
R          //-----------------------------------------------------------------
R
R          //-----------------------------------------------------------------
R          // Создание объекта LineLayoutFormPanel для имени и электронного адреса
R          LineLayoutFormPanel line2 = new LineLayoutFormPanel();
R
R          // Добавление к форму элементов для ввода имени
R                   line2.addElement(new LabelFormElement("Имя:"));
R          line2.addElement(user);
R          // Добавление к форме элементов для ввода электронного адреса
R          line2.addElement(new LabelFormElement("Электронный адрес:"));
R          line2.addElement(email);
R          line2.addElement(img);
R          //-----------------------------------------------------------------
R
```

```
R          //-------------------------------------------------------------------
R          // Создание разметки строк для вопросов
R          LineLayoutFormPanel line3 = new LineLayoutFormPanel();
R
R          // Добавление к форме элементов
R          line3.addElement(new LineLayoutFormPanel());
R          line3.addElement(new
R            CheckboxFormInput("use",
R "yes",
R                              "Используете ли вы Toolbox?",
R                              false));
R          line3.addElement(new LineLayoutFormPanel());
R          line3.addElement(new CheckboxFormInput(
R             "contact",
R             "yes",
R             "Would you like information about future Toolbox releases?",
R             true));
R          line3.addElement(new LineLayoutFormPanel());
R          //-------------------------------------------------------------------
R
R          //-------------------------------------------------------------------
R          // Создание группы радиокнопок для выбора версии
R          RadioFormInputGroup group = new RadioFormInputGroup("version");
R
R          // Добавление различных вариантов в эту группу
R          group.add(new RadioFormInput("version", "v3r2", "V3R2", false));
R          group.add(new RadioFormInput("version", "v4r1", "V4R1", false));
R          group.add(new RadioFormInput("version", "v4r2", "V4R2", false));
R          group.add(new RadioFormInput("version", "v4r3", "V4R3", false));
R          group.add(new RadioFormInput("version", "v4r4", "V4R4", false));
R          group.add(new
R            RadioFormInput("version",
R                           "multiple versions",
R                           "Несколько версий? А именно:",
R false));
R
R          // Создание элемента, допускающего выбор нескольких значений
R          SelectFormElement mlist = new SelectFormElement("MultiList");
R          mlist.setMultiple(true);
R          mlist.setSize(3);
R
R          // Добавление нескольких вариантов к предыдущему элементу
R          SelectOption option1 = mlist.addOption("V3R2", "v3r2");
R          SelectOption option2 = mlist.addOption("V4R1", "v4r1");
R          SelectOption option3 = mlist.addOption("V4R2", "v4r2");
R          SelectOption option4 = mlist.addOption("V4R3", "v4r3");
R          SelectOption option5 = mlist.addOption("V4R4", "v4r4");
R
R          // Создание текста HTML
R          txt = new HTMLText("Текущий уровень сервера:");
R          txt.setSize(4);
R
R          // Создание табличной разметки
R          GridLayoutFormPanel grid1 = new GridLayoutFormPanel(3);
R
R          // Добавление к ней группы радиокнопок и переключателей
R          grid1.addElement(txt);
R          grid1.addElement(group);
R          grid1.addElement(mlist);
R          //-------------------------------------------------------------------
R
R          //-------------------------------------------------------------------
R          // Создание табличной разметки для области применения
R          GridLayoutFormPanel grid2 = new GridLayoutFormPanel(1);
R          txt = new HTMLText("Текущие проекты или область интересов: " +
R                             "(проверка всех возможных)");
R          txt.setSize(4);
```

```
R
R          // Добавление элементов к табличной разметке
R          grid2.addElement(new LineLayoutFormPanel());
R          grid2.addElement(txt);
R          // Создание и добавление переключателя к табличной разметке
R          grid2.addElement(new CheckboxFormInput("interest", "applications", "Приложения", true));
R grid2.addElement(new CheckboxFormInput("interest", "applets", "Аплеты", false));
R grid2.addElement(new CheckboxFormInput("interest", "servlets", "Сервлеты", false));
R //-----------------------------------------------------------------
R
R          //-----------------------------------------------------------------
R          // Создание разметки строк для платформ
R          LineLayoutFormPanel line4 = new LineLayoutFormPanel();
R          txt = new HTMLText("Использованные клиентские платформы: " +
R                             "(проверка всех возможных)");
R          txt.setSize(4);
R
R          // Добавление элементов к разметке строк
R          line4.addElement(new LineLayoutFormPanel());
R          line4.addElement(txt);
R          line4.addElement(new LineLayoutFormPanel());
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "95",
R                                                 "Windows95",
R                                                 false));
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "98",
R                                                 "Windows98",
R                                                 false));
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "NT",
R                                                 "WindowsNT",
R                                                 false));
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "OS2",
R                                                 "OS/2",
R                                                 false));
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "AIX",
R                                                 "AIX",
R                                                 false));
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "Linux",
R                                                 "Linux",
R                                                 false));
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "AS400",
R                                                 "System i",
R                                                 false));
R          line4.addElement(new CheckboxFormInput("platform",
R                                                 "Other",
R                                                 "Другая:",
R                                                 false));
R
R          TextFormInput other = new TextFormInput("OtherPlatforms");
R          other.setSize(20);
R          other.setMaxLength(50);
R
R          line4.addElement(other);
R          //-----------------------------------------------------------------
R
R          //-----------------------------------------------------------------
R          // Создание поля для числа серверов
R          LineLayoutFormPanel grid3 = new LineLayoutFormPanel();
R
R          txt = new HTMLText(
R             "How many System i servers do you have?");
```

```
R           txt.setSize(4);
R
R           // Создание элемента для выбора числа серверов
R           SelectFormElement list = new SelectFormElement("list1");
R           // Создание и добавление к этому элементу различных вариантов
R           SelectOption opt0 = list.addOption("0", "нет");
R           SelectOption opt1 = list.addOption("1", "одна", true);
R           SelectOption opt2 = list.addOption("2", "две");
R           SelectOption opt3 = list.addOption("3", "три");
R           SelectOption opt4 = list.addOption("4", "четыре");
R           SelectOption opt5 = new SelectOption("5+", "пять или более", false);
R           list.addOption(opt5);
R
R           // Добавление элементов к табличной разметке
R           grid3.addElement(new LineLayoutFormPanel());
R           grid3.addElement(txt);
R           grid3.addElement(list);
R           //---------------------------------------------------------------------
R
R           //---------------------------------------------------------------------
R           // Создание табличной разметки для комментариев к продукту
R           GridLayoutFormPanel grid4 = new GridLayoutFormPanel(1);
R            txt = new HTMLText("Комментарии к продукту:");
R           txt.setSize(4);
R
R           // Добавление элементов к табличной разметке
R           grid4.addElement(new LineLayoutFormPanel());
R           grid4.addElement(txt);
R           // Создание поля ввода для комментариев
R           grid4.addElement(new TextAreaFormElement("comments", 5, 75));
R           grid4.addElement(new LineLayoutFormPanel());
R           //---------------------------------------------------------------------
R
R           //---------------------------------------------------------------------
R           // Создание табличной разметки
R           GridLayoutFormPanel grid5 = new GridLayoutFormPanel(2);
R           txt = new HTMLText("Войти в систему?");
R           txt.setSize(4);
R
R           // Создание поля ввода и метки для имени системы.
R           TextFormInput sys = new TextFormInput("system");
R           LabelFormElement sysLabel = new LabelFormElement("Система:");
R
R           // Создание поля ввода и метки для имени пользователя.
R           TextFormInput uid = new TextFormInput("uid");
R           LabelFormElement uidLabel = new LabelFormElement("Имя пользователя");
R
R           // Создание поля ввода пароля и метки для пароля.
R           PasswordFormInput pwd = new PasswordFormInput("pwd");
R           LabelFormElement pwdLabel = new LabelFormElement("Пароль");
R
R           // Добавление созданных элементов к табличной разметке
R           grid5.addElement(sysLabel);
R           grid5.addElement(sys);
R           grid5.addElement(uidLabel);
R           grid5.addElement(uid);
R           grid5.addElement(pwdLabel);
R           grid5.addElement(pwd);
R           //---------------------------------------------------------------------
R
R           //---------------------------------------------------------------------
R           // Добавление в форму HTML различных созданных
R           // панелей в порядке, в котором они будут появляться
R            form.addElement(line2);
R           form.addElement(line3);
R           form.addElement(grid1);
R           form.addElement(grid2);
```

```
R           form.addElement(line4);
R           form.addElement(grid3);
R           form.addElement(grid4);
R           form.addElement(txt);
R           form.addElement(new LineLayoutFormPanel());
R           form.addElement(grid5);
R           form.addElement(new LineLayoutFormPanel());
R           form.addElement(
R              new HTMLText("Submit an attachment Here: <br />"));
R           // Добавление поля выбора файла к форме
R           form.addElement(new FileFormInput("myAttachment"));
R           form.addElement(new ButtonFormInput("button",
R                                               "НАЖМИТЕ!",
R                                               "test()"));
R           // Добавление пустой разметки строк к форме. Это приведет
R           // adds a line break <br /> to the form
R           form.addElement(new LineLayoutFormPanel());
R           form.addElement(new LineLayoutFormPanel());
R           form.addElement(new SubmitFormInput("submit", "Зарегистрироваться"));
R           form.addElement(new LineLayoutFormPanel());
R           form.addElement(new LineLayoutFormPanel());
R           form.addElement(new ResetFormInput("reset", "Сброс"));
R           // Добавление скрытого поля к форме
R           form.addElement(new
R             HiddenFormInput("copyright",
R                             "(C) Copyright IBM Corp. 1999, 1999"));
R           //--------------------------------------------------------------------
R
R           // Вывод всей формы в буфер
R           page.append(form.toString());
R
R        }
R        catch(Exception e)
R        {
R           e.printStackTrace();
R        }
R
R        // Вывод закрывающих тегов HTML в буфер
R        page.append("</BODY>\n");
R        page.append("</HTML>\n");
R
R        // Выдача созданной страницы
R        return page.toString();
R
R     }
R  }
```

Пример вывода класса HTML

These are some possible sample outputs you may get from running the HTML class example.

R • Имя клиента: дядя Федор
R Электронный адрес: fedor@prostokvashino.ru
R Работает с AS/400 Toolbox для Java: да
R Требуется дополнительная информация: да
R Применяемые версии: v4r2,v4r4
R Область текущего или будущего применения Java: приложения,сервлеты
R Платформы: NT,Linux
R Number of System i servers: three
R Комментарии: Toolbox применяется всем отделом разработки для создания
R приложений по заказам пользователей
R Вложение: U:\wiedrich\servlet\temp.html
R (C) Copyright IBM Corp. 1999, 1999

R • Имя клиента: Карлсон
R Электронный адрес: carlson@roof.sv

R Работает с AS/400 Toolbox для Java: да
R Версия AS400: v4r4
R Область текущего или будущего применения Java: сервлеты
R Платформы: OS2
R Number of System i servers: FiveOrMore
R (C) Copyright IBM Corp. 1999, 1999

R • Имя клиента: Винни-Пух
R Электронный адрес: pooh@home.com
R Требуется дополнительная информация: да
R Версия AS400: v4r2
R Область текущего или будущего применения Java: приложения
R Платформы: NT,Другие
R Другие платформы: Solaris
R Number of System i servers: one
R Комментарии: Это первая программа, в которой используется
R этот класс! Очень хорошо!
R (C) Copyright IBM Corp. 1999, 1999

R • Имя клиента: Незнайка
R Электронный адрес: neznayka@suncity.su
R Версия AS400: v4r2
R Number of System i servers: one
R (C) Copyright IBM Corp. 1999, 1999

Ссылки, связанные с данной

“Пример: Применение классов форм HTML” на стр. 603
The following IBM Toolbox for Java example shows you how to use the HTML form classes.

Пример: Применение классов HTMLTree

This example shows how to to build HTML and file trees using the IBM Toolbox for Java HTML package classes.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// В этом примере демонстрируется применение классов HTML из
// IBM Toolbox for Java для создания деревьев файлов.
//
////////////////////////////////////////////////////////////////////////////

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;

/**
```

```
 *  Пример использования классов HTMLTree и FileTreeElement в сервлете.
 **/
public class TreeNav extends HttpServlet
{
   public void init(ServletConfig config)
      throws ServletException
   {
      super.init(config);

      // Набором значков по умолчанию в Toolbox представлены расширенные и
      // сжатые объекты, а также документы класса HTMLTree. Для расширенной работы
      // с этими значками с Toolbox поставляются три файла .gif (expanded.gif, collapsed.gif, bullet.gif),
      // расположенные в файле jt400Servlet.jar. Браузеры не позволяют выполнять поиск
      // файлов .gif в файлах .jar и .zip, поэтому эти изображения необходимо
      // извлечь из файла .jar и поместить в соответствующий каталог Web-сервера
      // (по умолчанию это каталог /html). После этого необходимо удалить
      // комментарии строк кода и задать правильный каталог в
      // этих методах присвоения. Каталог может быть как относительным, так и полным.

      HTMLTreeElement.setExpandedGif("/images/expanded.gif");
      HTMLTreeElement.setCollapsedGif("/images/collapsed.gif");
      HTMLTreeElement.setDocGif("/images/bullet.gif");
   }

   /**
    *  Обработка запроса GET.
    *  В параметре req - запрос.
    *  В параметре res - ответ.
    **/
   public void doGet (HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
   {
      HttpSession session = req.getSession(true);
      HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

      // Если в сеансе не создано дерево файлов,
      // создать начальное дерево.
      if (fileTree == null)
         fileTree = createTree(req, resp, req.getRequestURI());

      // Передача объекту HTMLTree запроса сервлета.
      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");

      // Получение тега HTMLTree.
      out.println(fileTree.getTag());

      out.println("</body>\n");
      out.println("</html>\n");
      out.close();

      // Сохранение параметров дерева
      // для следующего сеанса.
      session.putValue("filetree", fileTree);
   }

   /**
    *  Обработка запроса POST.
    *  В параметре req - запрос.
    *  В параметре res - ответ.
    **/
```

```
public void doPost (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
{
   res.setContentType("text/html");
   ServletOutputStream out = res.getOutputStream();
}

/**
 *  Создание начального объекта HTMLTree.
 **/
private HTMLTree createTree(HttpServletRequest req, HttpServletResponse resp, String uri)
{
   // Создание объекта HTMLTree.
   HTMLTree tree = new HTMLTree(req);

   try
   {
      // Создание объекта URLParser.
      URLParser urlParser = new URLParser(uri);

      AS400 sys = new AS400(CPUStatus.systemName_, "javactl", "jteam1");

      // Создание объекта File и указание корневого каталога в IFS.
      IFSJavaFile root = new IFSJavaFile(sys, "/QIBM");

      // Создание фильтра и списка всех каталогов.
      DirFilter filter = new DirFilter();
      //File[] dirList = root.listFiles(filter);

      // Получение списка файлов, соответствующих фильтру.
      String[] list = root.list(filter);

      File[] dirList = new File[list.length];

      // Мы не можем использовать возможности JDK1.2, поскольку
      // в JVM большинства Web-серверов пакет JDK обновляется с задержкой.
      // Эффективнее всего создать объекты файлов с помощью метода
      // listFiles(filter) из JDK1.2, как показано ниже, а не
      // с помощью метода list(filter) с последующим преобразованием
      // полученного массива строк в соответствующий массив
      // объектов File.
      // 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++)
      {
         // Создание объектов FileTreeElement для всех каталогов списка.
         FileTreeElement node = new FileTreeElement(dirList[i]);

         // Создание объектов ServletHyperlink для значков развертывания/свертывания.
         ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
         //sl.setHttpServletResponse(resp);
         node.setIconUrl(sl);

         // Создание объекта ServletHyperlink для сервлета TreeList,
         // показывающего содержимое каталога FileTreeElement.
         ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
         tl.setTarget("list");

         // Если объекту ServletHyperlink не присвоено имя,
```

```
            // присвоить ему имя каталога.
            if (tl.getText() == null)
               tl.setText(dirList[i].getName());

            // Задание объекта TextUrl для FileTreeElement.
            node.setTextUrl(tl);

            // Добавление FileTreeElement к объекту HTMLTree.
            tree.addElement(node);
         }
      }
      catch (Exception e)
      {
        e.printStackTrace();
      }

      return tree;
   }

   public void destroy(ServletConfig config)
   {
      // никаких действий выполнять не нужно
   }

   public String getServletInfo()
   {
       return "FileTree Navigation";
   }
}
```

Пример: создание просматриваемого дерева IFS (документ 1 из 3)

This IBM Toolbox for Java example code, in conjunction with the code in the other two example files, displays an HTMLTree and FileListElement in a servlet.

Этот пример состоит из следующих файлов:

- FileTreeExample.java - служит для создания фреймов HTML и запуска сервлета
- TreeNav.java - служит для создания и управления деревом
- TreeList.java - показывает содержимое объектов, выбранных в классе TreeNav.java

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of using the IBM Toolbox for Java HTML package
// IBM Toolbox для Java для создания деревьев файлов.
//
////////////////////////////////////////////////////////////////////////////

import java.io.PrintWriter;
import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.HTMLMeta;


//
// An example of using frames to display an HTMLTree and FileListElement
// в сервлете.
//

public class FileTreeExample extends HttpServlet
{
```

```
   public void init(ServletConfig config)
      throws ServletException
   {
      super.init(config);
   }

   /**
    *  Обработка запроса GET.
    *  В параметре req - запрос.
    *  В параметре res - ответ.
    **/

   public void doGet (HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
   {
      resp.setContentType("text/html");

      // Создание двух фреймов. Первый, навигационный фрейм, содержит
// объект HTMLTree, состоящий из объектов FileTreeElement. Он
      // позволяет перемещаться по файловой системе. Во втором фрейме
// будет показано содержимое каталога, выбранного в первом фрейме.
      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();
   }

   /**
    *  Обработка запроса POST.
    *  В параметре req - запрос.
    *  В параметре res - ответ.
    **/

   public void doPost (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
      res.setContentType("text/html");
      ServletOutputStream out = res.getOutputStream();
   }

   public void destroy(ServletConfig config)
   {
      // никаких действий выполнять не нужно
   }

   public String getServletInfo()
   {
      return "FileTree Servlet";
   }
}
```

Пример: создание просматриваемого дерева IFS (документ 2 из 3)

Приведенный фрагмент кода в сочетании с кодом из двух других файлов примеров демонстрируют применение объектов HTMLTree и FileListElement в сервлете.

Этот пример состоит из следующих файлов:

- FileTreeExample.java - служит для создания фреймов HTML и запуска сервлета
- TreeNav.java - данный файл, служащий для создания и управления деревом
- TreeList.java - показывает содержимое объектов, выбранных в классе TreeNav.java

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// В этом примере демонстрируется применение классов HTML из
// IBM Toolbox for Java для создания деревьев файлов.
//
////////////////////////////////////////////////////////////////////////////

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;

//
// Пример применения классов HTMLTree и FileTreeElement
// в сервлете.
//

public class TreeNav extends HttpServlet
{
   private AS400 sys_;

   public void init(ServletConfig config)
      throws ServletException
   {
      super.init(config);

      // Создание объекта AS400.
      sys_ = new AS400("mySystem", "myUserID", "myPassword");

      // В IBM Toolbox for Java значки по умолчанию представляют расширенные и
      // сжатые объекты, а также документы в HTMLTree. В состав
      // IBM Toolbox for Java входят три файла .gif, соответствующие значкам
      // (expanded.gif, collapsed.gif, bullet.gif),
      //  которые расположены в файле jt400Servlet.jar. Браузеры не позволяют выполнять поиск
      // файлов .gif в файлах .jar и .zip, поэтому эти изображения необходимо
      // извлечь из файла .jar и поместить в соответствующий каталог Web-сервера (по умолчанию это
      // каталог /html). После этого необходимо изменить приведенные ниже строки примеров,
      // указав в методах задания соответствующий каталог. Каталог может быть как относительным,
      // так и полным.

      HTMLTreeElement.setExpandedGif("http://myServer/expanded.gif");
      HTMLTreeElement.setCollapsedGif("http://myServer/collapsed.gif");
      HTMLTreeElement.setDocGif("http://myServer/bullet.gif");
   }

   /**
    *  Обработка запроса GET.
    *  В параметре req - запрос.
    *  В параметре res - ответ.
    **/

   public void doGet (HttpServletRequest req, HttpServletResponse resp)
```

```
   throws ServletException, IOException
{
   // Сохранение состояния дерева в данных сеанса.
   HttpSession session = req.getSession(true);
   HTMLTree fileTree = (HTMLTree)session.getValue("filetree");

   // Если в сеансе не создано дерево файлов,
   // создать начальное дерево.
   if (fileTree == null)
      fileTree = createTree(req, resp, req.getRequestURI());

   // Передача объекту HTMLTree запроса сервлета.
   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");

   // Получение тега HTMLTree.
   out.println(fileTree.getTag());

   out.println("</body>\n");
   out.println("</html>\n");
   out.close();

   // Сохранение параметров дерева
   // для следующего сеанса.
   session.putValue("filetree", fileTree);
}

/**
 *  Обработка запроса POST.
 *  В параметре req - запрос.
 *  В параметре res - ответ.
 **/

public void doPost (HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException
{
   res.setContentType("text/html");
   ServletOutputStream out = res.getOutputStream();
}

/**
 *  Создание начального объекта HTMLTree.
 **/

private HTMLTree createTree(HttpServletRequest req,
                            HttpServletResponse resp, String uri)
{
   // Создание объекта HTMLTree.
   HTMLTree tree = new HTMLTree(req);

     try
     {
       // Создание объекта URLParser.
       URLParser urlParser = new URLParser(uri);

       // Создание объекта File и указание корневого каталога в IFS.
       IFSJavaFile root = new IFSJavaFile(sys_, "/QIBM");

       // Создание фильтра.
       DirFilter filter = new DirFilter();
```

```
          // Получение списка файлов, соответствующих фильтру.
          String[] list = root.list(filter);

          File[] dirList = new File[list.length];

          // Мы не можем использовать возможности JDK1.2, поскольку
          // большая часть JVM web-серверов поздно обновляет уровень
          // JDK. Наиболее эффективный способ создания файловых объектов -
// метод listFiles(filter) из JDK1.2, выполняющий те же действия,
          // что и следующий код, вызывающий метод list(filter), а затем
          // преобразующий полученный список строк в массив объектов
          // типа File.
          // 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++)
          {
            // Создание объектов FileTreeElement для всех каталогов списка.
            FileTreeElement node = new FileTreeElement(dirList[i]);

            // Создание объектов ServletHyperlink для значков развертывания/свертывания.
            ServletHyperlink sl = new ServletHyperlink(urlParser.getURI());
            sl.setHttpServletResponse(resp);
            node.setIconUrl(sl);

            // Создание объекта ServletHyperlink для сервлета TreeList,
            // показывающего содержимое каталога FileTreeElement.
            ServletHyperlink tl = new ServletHyperlink("/servlet/TreeList");
            tl.setTarget("list");

            // Если объекту ServletHyperlink не присвоено имя,
            // присвоить ему имя каталога.
            if (tl.getText() == null)
              tl.setText(dirList[i].getName());

            // Задание объекта TextUrl для FileTreeElement.
            node.setTextUrl(tl);

            // Добавление FileTreeElement к объекту HTMLTree.
            tree.addElement(node);
          }

          sys_.disconnectAllServices();
        }

        catch (Exception e)
        {
          e.printStackTrace();
        }

        return tree;
    }

    public void destroy(ServletConfig config)
    {
       // никаких действий выполнять не нужно
    }

    public String getServletInfo()
```

```
    {
        return "FileTree Navigation";
    }
}
```

Пример: создание просматриваемого дерева IFS (документ 3 из 3)

Приведенный фрагмент кода в сочетании с кодом из двух других файлов примеров демонстрируют применение объектов HTMLTree и FileListElement в сервлете.

Этот пример состоит из следующих файлов:

- FileTreeExample.java - служит для создания фреймов HTML и запуска сервлета
- TreeNav.java - служит для создания и управления деревом
- TreeList.java - данный файл, показывающий содержимое объектов, выбранных в классе TreeNav.java

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// В этом примере демонстрируется применение классов HTML из
// IBM Toolbox для Java для создания списков файлов.
//
////////////////////////////////////////////////////////////////////////////

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.*;

/**
 *  Пример применения класса FileListElement в сервлете.
 **/
public class TreeList extends HttpServlet
{

    private AS400 sys_;

    /**
     *  Обработка запроса GET.
     *  В параметре req - запрос.
     *  В параметре res - ответ.
     **/
    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("Истекает",
                                     "в понедельник, 2 января 1990 г. в 13:00:00 по Гринвичу"));
```

```
            out.println("<body>\n");

            // Если путь не пуст, это означает, что пользователь выбрал элемент
            // списка FileTreeElement во фрейме навигации.
            if (req.getPathInfo() != null)
            {
                // Создание объекта FileListElement, передающего системный объект AS400 и
                // запрос сервлета Http. Запрос содержит данные о полном
                // пути, необходимые для просмотра содержимого выбранного объекта FileTreeElement
                // (каталога).
                FileListElement fileList = new FileListElement(sys_, req);

                //  Можно также создать объект FileListElement с применением
                //  имени общего ресурса NetServer
                //  и пути к общему ресурсу.
                //  FileListElement fileList =
                      new FileListElement(sys_, req, "TreeShare",
                                          "/QIBM/ProdData/HTTP/Public/jt400");


                // Вывод содержимого FileListElement.
                out.println(fileList.list());
            }
            // Показать заголовок HTMLHeading, если объект FileTreeElement не выбран.
            else
            {
                HTMLHeading heading = new
                  HTMLHeading(1,"Пример списка файлов HTML");
                heading.setAlign(HTMLConstants.CENTER);

                out.println(heading.getTag());
            }

            out.println("</body>\n");
            out.println("</html>\n");
            out.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

    }

    /**
      *  Обработка запроса POST.
      *  В параметре req - запрос.
      *  В параметре res - ответ.
    **/
    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);

        // Создание объекта AS400.
        sys_ = new AS400("mySystem", "myUID", "myPWD");
    }
}
```

Пример: Применение классов HTMLTable

The following example shows you how the HTMLTable classes work.

```
// Создание объекта HTMLTable по умолчанию.
HTMLTable table = new HTMLTable();

// Настройка атрибутов таблицы
table.setAlignment(HTMLTable.CENTER);
table.setBorderWidth(1);

// Создание объекта HTMLTableCaption по умолчанию и содержимого таблицы.
HTMLTableCaption caption = new HTMLTableCaption();
caption.setElement("Баланс счетов заказчиков - 1 января 2000 года");

// Добавление названия к таблице.
table.setCaption(caption);

// Создание заголовков столбцов и добавление их к таблице.
HTMLTableHeader account_header = new HTMLTableHeader(new HTMLText("Счет"));
HTMLTableHeader name_header = new HTMLTableHeader(new HTMLText("Имя"));
HTMLTableHeader balance_header = new HTMLTableHeader(new HTMLText("Баланс"));

table.addColumnHeader(account_header);
table.addColumnHeader(name_header);
table.addColumnHeader(balance_header);

// Добавление строк к таблице. Каждой записи о заказчике соответствует одна строка таблицы.
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));

   // Добавление строки к таблице.
   table.addRow(row);
}
System.out.println(table.getTag());
```

Приведенный выше фрагмент программы на Java создает следующий код HTML:

```
<table align="center" border="1">
<caption>Customer Account Balances - January 1, 2000</caption>
<tr>
<th>ACCOUNT</th>
<th>NAME</th>
<th>BALANCE</th>
</tr>
<tr align="center">
<td>0000001</td>
<td>Customer1</td>
<td>100.00</td>
</tr>
<tr align="center">
<td>0000002</td>
<td>Customer2</td>
<td>200.00</td>
</tr>
<tr align="center">
<td>0000003</td>
```

```
<td>Customer3</td>
<td>550.00</td>
</tr>
</table>
```

Ниже показано, как описанная таблица будет выглядеть на Web-странице:

Таблица 3. Баланс счетов заказчиков - 1 января 2000 года

Счет	Заказчик	Баланс
0000001	Заказчик_1	100.00
0000002	Заказчик_2	200.00
0000003	Заказчик_3	550.00

Примеры кода на Языке описаний вызовов программ (PCML)

В следующих примерах Язык описаний вызовов программ (PCML) применяется для вызова API i5/OS. После описания каждого примера приведена ссылка на документ, содержащий исходный код PCML и программу на Java.

Примечание: Необходимые права доступа для разных примеров различаются, но могут включать специальные права доступа к объектам и специальные права доступа. Для запуска этих примеров необходимо войти в систему с пользовательским профайлом, у которого есть права доступа для выполнения следующих операций:

- Вызов API i5/OS, рассматриваемого в примере
- Доступ к запрашиваемой информации

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Простой пример получения данных

This example the PCML source and Java program needed to retrieve information about a user profile on the server. The API being called is the Retrieve User Information (QSYRUSRI) API.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Данный пример состоит из двух частей:

- Исходный код PCML для вызова QSYRUSRI
- Java program source for calling QSYRUSRI

Исходный код PCML для вызова QSYRUSRI

```
<pcml version="1.0">

<!-- PCML source for calling "Retrieve user Information" (QSYRUSRI) API -->

  <!-- Format USRI0150 - Other formats are available -->
  <struct name="usri0100">
    <data name="bytesReturned"              type="int"    length="4"  usage="output"/>
    <data name="bytesAvailable"             type="int"    length="4"  usage="output"/>
    <data name="userProfile"                type="char"   length="10" usage="output"/>
    <data name="previousSignonDate"         type="char"   length="7"  usage="output"/>
    <data name="previousSignonTime"         type="char"   length="6"  usage="output"/>
    <data                                   type="byte"   length="1"  usage="output"/>
    <data name="badSignonAttempts"          type="int"    length="4"  usage="output"/>
    <data name="status"                     type="char"   length="10" usage="output"/>
    <data name="passwordChangeDate"         type="byte"   length="8"  usage="output"/>
    <data name="noPassword"                 type="char"   length="1"  usage="output"/>
    <data                                   type="byte"   length="1"  usage="output"/>
    <data name="passwordExpirationInterval" type="int"    length="4"  usage="output"/>
    <data name="datePasswordExpires"        type="byte"   length="8"  usage="output"/>
    <data name="daysUntilPasswordExpires"   type="int"    length="4"  usage="output"/>
    <data name="setPasswordToExpire"        type="char"   length="1"  usage="output"/>
    <data name="displaySignonInfo"          type="char"   length="10" usage="output"/>
  </struct>

  <!-- Program QSYRUSRI and its parameter list for retrieving USRI0100 format -->
  <program name="qsyrusri" path="/QSYS.lib/QSYRUSRI.pgm">
    <data name="receiver"         type="struct"   struct="usri0100"   usage="output"/>
    <data name="receiverLength"   type="int"      length="4"          usage="input" />
    <data name="format"           type="char"     length="8"          usage="input"    init="USRI0100"/>
    <data name="profileName"      type="char"     length="10"         usage="input"    init="*CURRENT"/>
    <data name="errorCode"        type="int"      length="4"          usage="input"    init="0"/>
  </program>

</pcml>
```

Исходный код программы на Java, вызывающей 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;

// Пример вызова API Получить информацию о пользователе (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;        // Код возврата ProgramCallDocument.callProgram()
        String msgId, msgText;     // Сообщения, полученные от сервера
        Object value;              // Значение, возвращенное ProgramCallDocument.getValue()

        System.setErr(System.out);

        // Создание объекта AS400 путем вызова конструктора без параметров,
    // все параметры будут запрошены у пользователя
        as400System = new AS400();

        try
        {
            // Для просмотра отладочной информации удалите символы
      // комментария в следующей строке
```

```
            //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

            System.out.println("Начало примера программы PCML..");
            System.out.println("    Создание объекта ProgramCallDocument для API QSYRUSRI...");

            // Создание ProgramCallDocument
            // Первый параметр задает систему для подключения
            // Второй параметр содержит имя ресурса pcml. В данном примере
// двоичный файл PCML qsyrusri.pcml.ser или
            // исходный файл PCML qsyrusri.pcml должны быть указаны в переменной classpath.
            pcml = new ProgramCallDocument(as400System, "qsyrusri");

            // Настройка входных параметров. Для некоторых параметров
            // заданы значения по умолчанию. Их не нужно переопределять в программе на Java.
System.out.println("    Установка входных параметров...");
            pcml.setValue("qsyrusri.receiverLength",
                          new Integer((pcml.getOutputsize("qsyrusri.receiver"))));

            // Вызов API
            // Появится приглашение на вход в систему
            System.out.println("    Вызов API QSYRUSRI для запроса сведений
                        о пользователе, зарегистрированном в системе.");
            rc = pcml.callProgram("qsyrusri");

            // Код возврата false означает, что получены сообщения от сервера
            if(rc == false)
            {
                // Получение списка сообщений сервера
                AS400Message[] msgs = pcml.getMessageList("qsyrusri");

                // Запись сообщений в стандартный поток вывода
                for (int m = 0; m < msgs.length; m++)
                {
                    msgId = msgs[m].getID();
                    msgText = msgs[m].getText();
                    System.out.println("    " + msgId + " - " + msgText);
                }
                System.out.println("** Не удалось вызвать API QSYRUSRI.
См. предыдущие сообщения**");
                System.exit(0);
            }
            // Если код возврата равен true, вызов QSYRUSRI выполнен успешно
            // Запись некоторых результатов в стандартный вывод
            else
            {
                value = pcml.getValue("qsyrusri.receiver.bytesReturned");
                System.out.println("        Байтов получено:      " + value);
                value = pcml.getValue("qsyrusri.receiver.bytesAvailable");
                System.out.println("        Байтов доступно:     " + value);
                value = pcml.getValue("qsyrusri.receiver.userProfile");
                System.out.println("        Имя профайла:        " + value);
                value = pcml.getValue("qsyrusri.receiver.previousSignonDate");
                System.out.println("        Дата предыдущего входа в систему:" + value);
                value = pcml.getValue("qsyrusri.receiver.previousSignonTime");
                System.out.println("        Время предыдущего входа в систему:" + value);
            }
        }
        catch (PcmlException e)
        {
            System.out.println(e.getLocalizedMessage());
            e.printStackTrace();
            System.out.println("*** Не удалось вызвать API QSYRUSRI. ***");
            System.exit(0);
        }
```

```
        System.exit(0);
    } // Конец main()

}
```

Пример: Получение списка информации

This example shows the PCML source and Java program needed to retrieve a list of authorized users on a server. The API being called is the Open List of Authorized Users (QGYOLAUS) API. Пример иллюстрирует работу с массивом структур, полученным от программы сервера.

Данный пример состоит из двух частей:

- Исходный код PCML для вызова QGYOLAUS
- Java program source for calling QGYOLAUS

Исходный код PCML для вызова QGYOLAUS

```
<pcml version="1.0">

<!-- PCML source for calling "Open List of Authorized Users" (QGYOLAUS) API -->

  <!-- Format AUTU0150 - Other formats are available -->
  <struct name="autu0150">
    <data name="name"         type="char" length="10" />
    <data name="userOrGroup"  type="char" length="1"  />
    <data name="groupMembers" type="char" length="1"  />
    <data name="description"  type="char" length="50" />
  </struct>

  <!-- List information structure (common for "Open List" type APIs) -->
  <struct name="listInfo">
    <data name="totalRcds"    type="int"  length="4" />
    <data name="rcdsReturned" type="int"  length="4" />
    <data name="rqsHandle"    type="byte" length="4" />
    <data name="rcdLength"    type="int"  length="4" />
    <data name="infoComplete" type="char" length="1" />
    <data name="dateCreated"  type="char" length="7" />
    <data name="timeCreated"  type="char" length="6" />
    <data name="listStatus"   type="char" length="1" />
    <data                     type="byte" length="1" />
    <data name="lengthOfInfo" type="int"  length="4" />
    <data name="firstRecord"  type="int"  length="4" />
    <data                     type="byte" length="40" />
  </struct>

  <!-- Program QGYOLAUS and its parameter list for retrieving AUTU0150 format -->
  <program name="qgyolaus" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm" parseorder="listInfo receiver">
    <data   name="receiver"       type="struct" struct="autu0150" usage="output"
              count="listInfo.rcdsReturned" outputsize="receiverLength" />
    <data   name="receiverLength" type="int"    length="4"        usage="input" init="16384" />
    <data   name="listInfo"       type="struct" struct="listInfo" usage="output" />
    <data   name="rcdsToReturn"   type="int"    length="4"        usage="input" init="264" />
    <data   name="format"         type="char"   length="10"       usage="input" init="AUTU0150" />
    <data   name="selection"      type="char"   length="10"       usage="input" init="*USER" />
    <data   name="member"         type="char"   length="10"       usage="input" init="*NONE" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />

  </program>

  <!-- Программа QGYGTLE возвратила дополнительные записи из списка,
       созданного программой QGYOLAUS. -->
  <program name="qgygtle" path="/QSYS.lib/QGY.lib/QGYGTLE.pgm" parseorder="listInfo receiver">
    <data   name="receiver"       type="struct" struct="autu0150" usage="output"
            count="listInfo.rcdsReturned" outputsize="receiverLength" />
    <data   name="receiverLength" type="int"    length="4"        usage="input" init="16384" />
    <data   name="requestHandle"  type="byte"   length="4"        usage="input" />
```

```
    <data   name="listInfo"       type="struct" struct="listInfo" usage="output" />
    <data   name="rcdsToReturn"   type="int"    length="4"        usage="input" init="264" />
    <data   name="startingRcd"    type="int"    length="4"        usage="input" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />
  </program>

  <!-- Program QGYCLST closes the list, freeing resources on the server -->
  <program name="qgyclst" path="/QSYS.lib/QGY.lib/QGYCLST.pgm" >
    <data   name="requestHandle"  type="byte"   length="4"        usage="input" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />
  </program>
</pcml>
```

Исходный код программы на Java, вызывающей 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;

// Пример программы, вызывающей API Получить список пользователей с правами доступа (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;        // Код возврата ProgramCallDocument.callProgram()
    String msgId, msgText;     // Сообщения, полученные от сервера
    Object value;              // Значение, возвращенное ProgramCallDocument.getValue()

    int[] indices = new int[1]; // Индексы массива
    int nbrRcds,                // Число записей, возвращенных QGYOLAUS и QGYGTLE
        nbrUsers;               // Общее число полученных имен пользователей
    String listStatus;          // Состояние списка на сервере
    byte[] requestHandle = new byte[4];

    System.setErr(System.out);

    // Создание объекта AS400 путем вызова конструктора без параметров,
    // все параметры будут запрошены у пользователя
    as400System = new AS400();

    try
    {
      // Для просмотра отладочной информации удалите символы
      // комментария в следующей строке
      //com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Начало примера программы PCML..");
      System.out.println("    Создание объекта ProgramCallDocument для API QGYOLAUS...");

      // Создание ProgramCallDocument
      // Первый параметр задает систему для подключения
      // Второй параметр содержит имя ресурса pcml. В данном примере каталог
// двоичного файла PCML qgyolaus.pcml.ser или
      // исходного файла PCML qgyolaus.pcml должны быть указаны в переменной classpath.
      pcml = new ProgramCallDocument(as400System, "qgyolaus");

      // Для всех входных параметров в файле PCML заданы значения по умолчанию.
      // Их не нужно переопределять в программе на Java.

      // Вызов API
      // Появится приглашение на вход в систему
      System.out.println("    Вызов API QSYRUSRI для запроса сведений
               о пользователе, зарегистрированном в системе.");
```

```
      rc = pcml.callProgram("qgyolaus");

      // Код возврата false означает, что получены сообщения от сервера
      if(rc == false)
      {
        // Получение списка сообщений сервера
        AS400Message[] msgs = pcml.getMessageList("qgyolaus");

        // Запись сообщений в стандартный поток вывода
        for (int m = 0; m < msgs.length; m++)
        {
            msgId = msgs[m].getID();
            msgText = msgs[m].getText();
            System.out.println("    " + msgId + " - " + msgText);
        }
        System.out.println("** Не удалось вызвать API QGYOLAUS.
См. предыдущие сообщения**");
        System.exit(0);
      }
      // Если код возврата равен true, вызов QGYOLAUS выполнен успешно
      // Запись некоторых результатов в стандартный вывод
      else
      {
        boolean doneProcessingList = false;
        String programName = "qgyolaus";
        nbrUsers = 0;
        while (!doneProcessingList)
        {
          nbrRcds = pcml.getIntValue(programName + ".listInfo.rcdsReturned");
          requestHandle = (byte[]) pcml.getValue(programName + ".listInfo.rqsHandle");

          // Цикл по списку пользователей
          for (indices[0] = 0; indices[0] < nbrRcds; indices[0]++)
          {
               value = pcml.getValue(programName + ".receiver.name", indices);
               System.out.println("Пользователь:  " + value);

               value = pcml.getValue(programName + ".receiver.description", indices);
               System.out.println("\t\t" + value);
          }

          nbrUsers += nbrRcds;

          // Проверка, все ли пользователи получены.
          // Если получены сведения не обо всех пользователях, API Получить записи списка (QGYGTLE)
          // будет дополнительно вызван один или несколько раз для получения остальных записей списка.
          listStatus = (String) pcml.getValue(programName + ".listInfo.listStatus");
          if ( listStatus.equals("2")   // Список помечен как полный,
            || listStatus.equals("3") ) // либо как содержащий ошибки
          {
            doneProcessingList = true;
          }
          else
          {
            programName = "qgygtle";

            // Указание входных параметров QGYGTLE
            pcml.setValue("qgygtle.requestHandle", requestHandle);
            pcml.setIntValue("qgygtle.startingRcd", nbrUsers + 1);

            // Получение дополнительных записей списка пользователей
            // с помощью API Получить записи списка (QGYGTLE)
            rc = pcml.callProgram("qgygtle");

            // Код возврата false означает, что получены сообщения от сервера
            if(rc == false)
            {
```

```
                // Получение списка сообщений сервера
                AS400Message[] msgs = pcml.getMessageList("qgygtle");

                // Запись сообщений в стандартный поток вывода
                for (int m = 0; m < msgs.length; m++)
                {
                    msgId = msgs[m].getID();
                    msgText = msgs[m].getText();
                    System.out.println("    " + msgId + " - " + msgText);
                }
                System.out.println("** Не удалось вызвать API QGYGTLE.
См. предыдущие сообщения**");
                System.exit(0);
              }
              // Если код возврата равен true, вызов QGYGTLE выполнен успешно

            }
          }
          System.out.println("Число возвращенных записей с информацией о пользователях:  " + nbrUsers);

          // Вызов API Закрыть список (QGYCLST)
          pcml.setValue("qgyclst.requestHandle", requestHandle);
          rc = pcml.callProgram("qgyclst");
        }
      }
      catch(PcmlException e)
      {
        System.out.println(e.getLocalizedMessage());
        e.printStackTrace();
        System.out.println("** Не удалось вызвать API QGYOLAUS. ***");
        System.exit(0);
      }

      System.exit(0);
    }
}
```

Пример: Получение многомерных данных

This example show the PCML source and Java program needed to retrieve a list Network File System (NFS) exports from a server. The API being called is the Retrieve NFS Exports (QZNFRTVE) API. Пример иллюстрирует работу с массивом структур, вложенным в другой массив структур.

Данный пример состоит из двух частей:

- Исходный код PCML для вызова QZNFRTVE
- Java program source for calling QZNFRTVE

Исходный код PCML для вызова 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>
```

Исходный код программы на Java, вызывающей 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;

// Пример программы, вызывающей API Получить экспортированные объекты 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;        // Код возврата ProgramCallDocument.callProgram()
    String msgId, msgText;     // Сообщения, полученные от сервера
    Object value;              // Значение, возвращенное ProgramCallDocument.getValue()

    System.setErr(System.out);

    // Создание объекта AS400 путем вызова конструктора без параметров,
    // все параметры будут запрошены у пользователя
    as400System = new AS400();

    int[] indices = new int[2]; // Индексы массива
    int nbrExports;             // Полученное число экспортируемых файлов
    int nbrOfReadWriteHostNames, nbrOfRWHostNames, nbrOfRootHostNames,
        nbrOfAccessHostnames,    nbrOfHostOpts;

    try
    {
      // Для просмотра отладочной информации удалите символы
      // комментария в следующей строке
      // com.ibm.as400.data.PcmlMessageLog.setTraceEnabled(true);

      System.out.println("Начало примера программы PCML..");
      System.out.println("    Создание объекта ProgramCallDocument для API QZNFRTVE...");

      // Создание ProgramCallDocument
      // Первый параметр задает систему для подключения
      // Второй параметр содержит имя ресурса pcml. В данном примере
// каталоги двоичного файла PCML qznfrtve.pcml.ser или исходного
      // файла PCML qznfrtve.pcml должны быть указаны в переменной classpath.
      pcml = new ProgramCallDocument(as400System, "qznfrtve");

      // Настройка входных параметров. Для некоторых параметров
      // заданы значения по умолчанию. Их не нужно переопределять в программе на Java.
System.out.println("    Установка входных параметров...");
      pcml.setValue("qznfrtve.receiverLength", new Integer( ( pcml.getOutputsize("qznfrtve.receiver"))));

      // Вызов API
      // Появится приглашение на вход в систему
      System.out.println("    Вызов API QZNFRTVE, отправляющего запрос на объекты NFS.");
      rc = pcml.callProgram("qznfrtve");

      if (rc == false)
      {
        // Получение списка сообщений сервера
        AS400Message[] msgs = pcml.getMessageList("qznfrtve");

        // Запись сообщений в стандартный поток вывода
        for (int m = 0; m < msgs.length; m++)
        {
            msgId = msgs[m].getID();
            msgText = msgs[m].getText();
            System.out.println("    " + msgId + " - " + msgText);
        }
        System.out.println("** Не удалось вызвать API QZNFRTVE. См. предыдущие сообщения**");
```

```
        System.exit(0);
      }
      // Если код возврата равен true, вызов QZNFRTVE выполнен успешно
      // Запись некоторых результатов в стандартный вывод
      else
      {
        nbrExports = pcml.getIntValue("qznfrtve.returnedRcdsFdbkInfo.nbrOfNFSExportEntries");
        // Вывод элементов списка
        for (indices[0] = 0; indices[0] < nbrExports; indices[0]++)
        {
          value = pcml.getValue("qznfrtve.receiver.pathName", indices);
          System.out.println("Полное имя = " + value);

          // Вывод имен хостов для чтения-записи
          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("    Имя хоста для чтения/записи = " + value);
          }

          // Вывод имени корневого хоста
          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("    Имя хоста корневого каталога = " + value);
          }

          // Вывод имен хостов для доступа
          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("    Имя хоста доступа = " + value);
          }

          // Вывод опций хоста
          nbrOfHostOpts = pcml.getIntValue("qznfrtve.receiver.nbrOfHostOptions", indices);
          for(indices[1] = 0; indices[1] < nbrOfHostOpts; indices[1]++)
          {
            System.out.println("    Параметры хоста:");
            value = pcml.getValue("qznfrtve.receiver.hostOptions.dataFileCodepage", indices);
            System.out.println("        Кодовая страница файла данных = " + value);
            value = pcml.getValue("qznfrtve.receiver.hostOptions.pathNameCodepage", indices);
            System.out.println("        Кодовая страница полного имени = " + value);
            value = pcml.getValue("qznfrtve.receiver.hostOptions.writeModeFlag", indices);
            System.out.println("        Флаг режима записи = " + value);
            value = pcml.getValue("qznfrtve.receiver.hostOptions.hostName", indices);
            System.out.println("        Имя хоста = " + value);
          }
        } // Конец цикла по списку экспортируемых файлов
      } // Завершение обработки успешного вызова QZNFRTVE
    }
    catch(PcmlException e)
    {
      System.out.println(e.getLocalizedMessage());
      e.printStackTrace();
      System.exit(-1);
    }

    System.exit(0);
  } // Конец main()
}
```

Примеры: Классы ReportWriter

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java ReportWriter classes.

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Работа с классом JSPReportProcessor с помощью PDFContext

This example uses the JSPReportProcessor and the PDFContext classes to obtain data from a specified URL and convert the data to the PDF format. The data is then streamed to a file as a PDF document.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

Пример программы JSP, который можно применять при работе с классом JSPRunReport, можно просмотреть в файле JSPcust_table.jsp. Вы можете также загрузить архивный файл ZIP, который содержит примеры программы для JSP. Этот файл содержит также примеры для XML and XSL, которые можно применять с примером XSLReportProcessor (PCLRunReport).

```
////////////////////////////////////////////////////////////////////////////
//
// В этом примере (JSPRunReport) демонстрируется применение классов JSPReportProcessor
// и PDFContext для получения данных, расположенных по указанному адресу, и их
// преобразования в формат PDF. Данные записываются в файл в виде документа PDF.
//
// Формат вызова:
//     java JSPRunReport <jsp_Url> <output_filename>
//
////////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.awt.print.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.HashMap;

import com.ibm.xsl.composer.flo.*;
import com.ibm.xsl.composer.areas.*;
import com.ibm.xsl.composer.framework.*;
import com.ibm.xsl.composer.java2d.*;
import com.ibm.xsl.composer.prim.*;
import com.ibm.xsl.composer.properties.*;
import com.ibm.as400.util.reportwriter.processor.*;
import com.ibm.as400.util.reportwriter.pdfwriter.*;
import java.io.IOException;
import java.io.Serializable;
import org.xml.sax.SAXException;
```

```
public class JSPRunReport

{

    public static void main( String args[] )
    {
        FileOutputStream fileout = null;

        /** указание URL, содержащего данные, которые необходимо использовать при создании отчета **/
        String JSPurl = args[0];
        URL jspurl = null;
        try {
              jspurl = new URL(JSPurl);
        }
        catch (MalformedURLException e)
        {}

        /** получение файла вывода PDML  **/
        String filename = args[1];
        try {
        fileout = new FileOutputStream(filename);
        }
        catch (FileNotFoundException e)
        {}

        /** настройка формата страницы **/
        Paper paper = new Paper();
        paper.setSize(612,792);
        paper.setImageableArea(18, 18, 576, 756);
        PageFormat pf = new PageFormat();
        pf.setPaper(paper);

        /** создание объекта PDFContext и указание FileOutputStream в качестве OutputStream **/
        PDFContext pdfcontext = new PDFContext((OutputStream)fileout, pf);

        System.out.println( Ready to parse XSL document );

        /** создание объекта JSPReportProcessor и установка шаблона для выбранного JSP **/
        JSPReportProcessor jspprocessor = new JSPReportProcessor(pdfcontext);
        try {
        jspprocessor.setTemplate(jspurl);
        }

        catch (NullPointerException np){
            String mes = np.getMessage();
            System.out.println(mes);
            System.exit(0);
            }

        /** обработка отчета **/
        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);
    }
}
```

Пример: Файл JSP для класса JSPReportProcessor

Use this sample file with the Example: Using JSPReportProcessor with PDFContext topic.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
<?xml version="1.0"?>

<!--
  Copyright (c) 1999 The Apache Software Foundation.  All rights reserved.
-->

<%@ page session="false"%>
<%@ page language="java" contentType="text/html" %>
<%@ page import="java.lang.*" %>
<%@ page import="java.util.*" %>


<%-- <jsp:useBean id='cust_table' scope='page' class='table.JSPcust_table'/> --%>

<%!
   String[][] cust_data = new String [4][5];

   public void jspInit()
   {
     //cust_record_field [][] cust_data;
     // cust_record содержит имя, адрес, название города и штата, а также
     // zip-код заказчика


     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;
   }
%>

<!-- Первый тест анализатора и средств создания. -->
<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>

   <%
     // добавление строки в таблицу
     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>

<%
} // конец строки while
%>
```

```
        </fo:table-body>
      </fo:table>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Пример: Применение XSLReportProcessor с PCLContext

This example should not be used as the XSLReportProcessor class is no longer supported.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
R ////////////////////////////////////////////////////////////////////////////
R //
R // В следующем примере (PCLRunReport) классы XSLPReportProcessor и
R // PCLContext применяются для получения данных XML и их преобразования в формат PCL.
R // Затем данные отправляются на принтер OutputQueue.
R //
R // Содержимое примеров исходных файлов XML и XSL, которые можно применять
R // с PCLRunReport, можно просмотреть в realestate.xml и realestate.xsl.
R // Вы можете также загрузить ZIP-файл с примерами файлов XML и XSL. Кроме того,
R // ZIP-файл содержит пример файла JSP, который можно применять
R // с примером класса JSPReportProcessor (JSPRunReport).
R //
R // Формат вызова:
R //     java PCLRunReport <xml_file> <xsl_file>
R //
R ////////////////////////////////////////////////////////////////////////////
R
R import java.lang.*;
R import java.awt.*;
R import java.io.*;
R import java.awt.print.*;
R import java.awt.event.*;
R import java.util.LinkedList;
R import java.util.ListIterator;
R import java.util.HashMap;
R
R import com.ibm.xsl.composer.flo.*;
R import com.ibm.xsl.composer.areas.*;
R import com.ibm.xsl.composer.framework.*;
R import com.ibm.xsl.composer.java2d.*;
R import com.ibm.xsl.composer.prim.*;
R import com.ibm.xsl.composer.properties.*;
R import com.ibm.as400.util.reportwriter.processor.*;
R import com.ibm.as400.util.reportwriter.pclwriter.*;
R import java.io.IOException;
R import java.io.Serializable;
R import org.xml.sax.SAXException;
R import com.ibm.as400.access.*;
R
R public class PCLRunReport
R
R {
R
R     public static void main( String args[] )
R     {
R         SpooledFileOutputStream fileout = null;
R         String xmldocumentName = args[0];
R         String xsldocumentName = args[1];
R
R         String sys = "<system>";      /* Insert server name       */
R         String user = "<user>";       /* Insert user profile name */
R         String pass = "<password>";   /* Insert password          */
R
```

```
R         AS400 system = new AS400(sys, user, pass);
R
R         /* Insert output queue */
R         String outqname = "/QSYS.LIB/qusrsys.LIB/<outq>.OUTQ";
R         OutputQueue outq = new OutputQueue(system, outqname);
R         PrintParameterList parms = new PrintParameterList();
R         parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outq.getPath());
R
R         try{
R           fileout = new SpooledFileOutputStream(system, parms, null, null);
R         }
R         catch (Exception e)
R         {}
R
R         /** настройка формата страницы **/
R         Paper paper = new Paper();
R         paper.setSize(612,792);
R         paper.setImageableArea(18, 36, 576, 720);
R         PageFormat pf = new PageFormat();
R         pf.setPaper(paper);
R
R         /** создание объекта PCLContext и указание FileOutputStream
R             в качестве объекта OutputStream **/
R         PCLContext pclcontext = new PCLContext((OutputStream)fileout, pf);
R
R         System.out.println("Программа готова к анализу документа XSL");
R
R         /** создание объекта XSLReportProcessor **/
R         XSLReportProcessor xslprocessor = new XSLReportProcessor(pclcontext);
R         try {
R         xslprocessor.setXMLDataSource(xmldocumentName);
R         }
R         catch (SAXException se) {
R             String mes = se.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R         catch (IOException ioe) {
R             String mes = ioe.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R         catch (NullPointerException np){
R             String mes = np.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R         /** настройка шаблона согласно указанному источнику данных XML **/
R         try {
R         xslprocessor.setTemplate(xsldocumentName);
R         }
R         catch (NullPointerException np){
R             String mes = np.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R         catch (IOException e) {
R             String mes = e.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R         catch (SAXException se) {
R             String mes = se.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R
```

```
R         /** обработка отчета **/
R         try {
R         xslprocessor.processReport();
R         }
R         catch (IOException e) {
R             String mes = e.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R         catch (SAXException se) {
R             String mes = se.getMessage();
R             System.out.println(mes);
R             System.exit(0);
R             }
R
R         System.exit(0);
R     }
R
R }
```

Пример: Пример файла XML для класса XSLReportProcessor

This example should not be used as the XSLReportProcessor class is no longer supported.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
<?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>
  То, о чем вы могли только мечтать.
     </DISCLOSURES>
  <UTILITIES>
     Да
  </UTILITIES>
```

```
 <EXTRAS>
    Защита от насекомых.
 </EXTRAS>
 <CONSTRUCTION>
    Стеновые панели и клей
 </CONSTRUCTION>
 <ACCESS>
    Входная дверь.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
    Да.
 </ASSUMABLE>
 <OWNER-CARRY>
    Слишком тяжелый.
 </OWNER-CARRY>
 <ASSESMENTS>
    $150000
 </ASSESMENTS>
 <DUES>
    $100
 </DUES>
 <TAXES>
    $2000
 </TAXES>
 <LENDER>
    Есть закладная.
 </LENDER>
 <EARNEST>
    Берт
 </EARNEST>
 <DIRECTIONS>
    Север, юг, восток, запад
 </DIRECTIONS>
 </FINANCIAL>

 <REMARKS>
 </REMARKS>

 <CONTACTS>
<COMPANY>
 <NAME>
    Надежная недвижимость
 </NAME>
 <ADDRESS>
    Главная улица, 12
 </ADDRESS>
 <CITY>
    Ловелл, MA
 </CITY>
 <ZIP>
    34567
 </ZIP>
</COMPANY>
<AGENT>
 <NAME>
    Мэри Джонс
 </NAME>
 <ADDRESS>
 </ADDRESS>
 <CITY>
 </CITY>
 <ZIP>
 </ZIP>
</AGENT>
```

```
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Да.
 </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>
    Мэри, продавец недвижимости
   </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>
  Дом
 </TYPE>

 <PRICE>
  $200000
 </PRICE>

        <AGE UNITS="MONTHS">
  3
 </AGE>

        <LOCATION COUNTRY="USA" STATE="CO" COUNTY="MIDDLESEX" SECURITY="Public">
  <ADDRESS>
   1 Главная улица
  </ADDRESS>
```

```
 <CITY>
     Бэлдер
 </CITY>
 <ZIP>
  11111
 </ZIP>
</LOCATION>

<STRUCTURE>
  <NUM-BEDS>
    2
          </NUM-BEDS>
  <NUM-BATHS>
    2
          </NUM-BATHS>
</STRUCTURE>

<DATES>
 <LISTING-DATE>
     3.4.98
                </LISTING-DATE>
</DATES>

       <LAND-AREA UNITS="ACRES">
 0,01
</LAND-AREA>

 </GENERAL>

 <FEATURES>
    <DISCLOSURES>
 То, о чем вы могли только мечтать.
    </DISCLOSURES>
 <UTILITIES>
    Да
 </UTILITIES>
 <EXTRAS>
    Защита от насекомых.
 </EXTRAS>
 <CONSTRUCTION>
    Стеновые панели и клей
 </CONSTRUCTION>
 <ACCESS>
    Входная дверь.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
    Да.
 </ASSUMABLE>
 <OWNER-CARRY>
    Слишком тяжелый.
 </OWNER-CARRY>
 <ASSESMENTS>
    $150000
 </ASSESMENTS>
 <DUES>
    $100
 </DUES>
 <TAXES>
    $2000
 </TAXES>
 <LENDER>
    Есть закладная.
 </LENDER>
 <EARNEST>
```

```
    Берт
  </EARNEST>
  <DIRECTIONS>
    Север, юг, восток, запад
  </DIRECTIONS>
  </FINANCIAL>

  <REMARKS>
  </REMARKS>

  <CONTACTS>
 <COMPANY>
  <NAME>
    Надежная недвижимость
  </NAME>
  <ADDRESS>
    Главная улица, 12
  </ADDRESS>
  <CITY>
    Ловелл, MA
  </CITY>
  <ZIP>
    34567
  </ZIP>
 </COMPANY>
 <AGENT>
  <NAME>
    Мэри Джонс
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </AGENT>
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Да.
 </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>
```

```
   Боб, продавец недвижимости
  </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>
 Квартира
</TYPE>

<PRICE>
 $65000
</PRICE>

       <AGE UNITS="YEARS">
 30
</AGE>

       <LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
 <ADDRESS>
  Вишневая улица, 25.
 </ADDRESS>
 <CITY>
     Кэмбридж
 </CITY>
 <ZIP>
  02139
 </ZIP>
</LOCATION>

<STRUCTURE>
  <NUM-BEDS>
    3
         </NUM-BEDS>
  <NUM-BATHS>
    1
         </NUM-BATHS>
</STRUCTURE>

<DATES>
 <LISTING-DATE>
     5.3.97
               </LISTING-DATE>
</DATES>

       <LAND-AREA UNITS="ACRES">
 0.05
</LAND-AREA>

 </GENERAL>

 <FEATURES>
    <DISCLOSURES>
 То, о чем вы могли только мечтать.
```

```
   </DISCLOSURES>
 <UTILITIES>
   Да
 </UTILITIES>
 <EXTRAS>
   Защита от насекомых.
 </EXTRAS>
 <CONSTRUCTION>
   Стеновые панели и клей
 </CONSTRUCTION>
 <ACCESS>
   Входная дверь.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
   Да.
 </ASSUMABLE>
 <OWNER-CARRY>
   Слишком тяжелый.
 </OWNER-CARRY>
 <ASSESMENTS>
   $150000
 </ASSESMENTS>
 <DUES>
   $100
 </DUES>
 <TAXES>
   $2000
 </TAXES>
 <LENDER>
   Есть закладная.
 </LENDER>
 <EARNEST>
   Берт
 </EARNEST>
 <DIRECTIONS>
   Север, юг, восток, запад
 </DIRECTIONS>
 </FINANCIAL>

 <REMARKS>
 </REMARKS>

 <CONTACTS>
<COMPANY>
 <NAME>
   Надежная недвижимость
 </NAME>
 <ADDRESS>
   Главная улица, 12
 </ADDRESS>
 <CITY>
   Ловелл, MA
 </CITY>
 <ZIP>
   34567
 </ZIP>
</COMPANY>
<AGENT>
 <NAME>
   Мэри Джонс
 </NAME>
 <ADDRESS>
 </ADDRESS>
 <CITY>
```

```
  </CITY>
  <ZIP>
  </ZIP>
 </AGENT>
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Да.
 </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>
    Мэри, продавец недвижимости
   </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>
       Дом
 </TYPE>

 <PRICE>
  $449000
 </PRICE>

       <AGE UNITS="YEARS">
  7
 </AGE>

       <LOCATION COUNTRY="USA" STATE="MA" COUNTY="MIDDLESEX" SECURITY="Public">
```

```
 <ADDRESS>
  Западное шоссе, 100
 </ADDRESS>
 <CITY>
     Лексингтон
 </CITY>
 <ZIP>
  02421
 </ZIP>
</LOCATION>

<STRUCTURE>
  <NUM-BEDS>
    7
          </NUM-BEDS>
  <NUM-BATHS>
    3
          </NUM-BATHS>
</STRUCTURE>

<DATES>
 <LISTING-DATE>
     8.6.98
                </LISTING-DATE>
</DATES>

       <LAND-AREA UNITS="ACRES">
 2,0
</LAND-AREA>

 </GENERAL>

 <FEATURES>
    <DISCLOSURES>
 То, о чем вы могли только мечтать.
    </DISCLOSURES>
 <UTILITIES>
    Да
 </UTILITIES>
 <EXTRAS>
    Защита от насекомых.
 </EXTRAS>
 <CONSTRUCTION>
    Стеновые панели и клей
 </CONSTRUCTION>
 <ACCESS>
    Входная дверь.
 </ACCESS>
 </FEATURES>

 <FINANCIAL>
 <ASSUMABLE>
    Да.
 </ASSUMABLE>
 <OWNER-CARRY>
    Слишком тяжелый.
 </OWNER-CARRY>
 <ASSESMENTS>
    $300000
 </ASSESMENTS>
 <DUES>
    $100
 </DUES>
 <TAXES>
    $2000
 </TAXES>
 <LENDER>
```

```
    Есть закладная.
  </LENDER>
  <EARNEST>
    Берт
  </EARNEST>
  <DIRECTIONS>
    Север, юг, восток, запад
  </DIRECTIONS>
  </FINANCIAL>

  <REMARKS>
  </REMARKS>

  <CONTACTS>
 <COMPANY>
  <NAME>
    Надежная недвижимость
  </NAME>
  <ADDRESS>
    Главная улица, 12
  </ADDRESS>
  <CITY>
    Ловелл, MA
  </CITY>
  <ZIP>
    34567
  </ZIP>
 </COMPANY>
 <AGENT>
  <NAME>
    Мэри Джонс
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </AGENT>
 <OWNER>
  <NAME>
  </NAME>
  <ADDRESS>
  </ADDRESS>
  <CITY>
  </CITY>
  <ZIP>
  </ZIP>
 </OWNER>
    <TENANT>
    Да.
 </TENANT>
    <COMMISION>
    15%
 </COMMISION>
  </CONTACTS>

</RESIDENTIAL-LISTING>

</RESIDENTIAL-LISTINGS>
```

Пример: Пример файла XSL для класса XSLReportProcessor

This example should not be used as the XSLReportProcessor class is no longer supported.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
<?xml version="1.0"?>

<!-- Образец оформления документа, фиксирующего сделку с недвижимостью. -->
<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>
```

Примеры: Классы ресурсов

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java resource classes.

Resource и ChangeableResource

- Пример: получение значения атрибута от RUser, действительного подкласса Resource
- Пример: Установка значений атрибутов для RJob, действительного подкласса ChangeableResource
- Пример: Доступ к ресурсам с помощью общего кода

ResourceList

- Пример: Получение и печать содержимого ResourceList
- Пример: Обращение к объекту ResourceList с помощью общего кода
- Пример: Представление списка ресурсов сервлета

Presentation

- Пример: Применение объектов Presentation

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Примеры: Список ресурсов

The following examples show various ways of working with resource lists.

- Пример: Получение и печать содержимого объекта ResourceList
- Пример: Обращение к ResourceList с помощью общего кода
- Пример: Представление списка ресурсов в сервлете

Пример: Получение и печать содержимого объекта ResourceList

R One example of a concrete subclass of ResourceList is com.ibm.as400.resource.RJobList, which represents a list of
R system jobs. RJobList supports many selection IDs and sort IDs, each of which can be used to filter or sort the list.This
R example prints the contents of an RJobList:

```
// Создать объект RJobList, который будет представлять список заданий.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJobList jobList = new RJobList(system);

// Отфильтровать список для просмотра только интерактивных заданий.
jobList.setSelectionValue(RJobList.JOB_TYPE, RJob.JOB_TYPE_INTERACTIVE);

// Сортировать список по имени пользователя, затем по имени задания.
Object[] sortValue = new Object[] { RJob.USER_NAME, RJob.JOB_NAME };
jobList.setSortValue(sortValue);

// Открыть список и дождаться завершения его составления.
jobList.open();
jobList.waitForComplete();

// Считать и отобразить содержимое списка.
long length = jobList.getListLength();
for(long i = 0; i < length; ++i)
{
    System.out.println(jobList.resourceAt(i));
}

// Закрыть список.
jobList.close();
```

Отказ от гарантий на предоставляемый код

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Информация, связанная с данной

RJobList Javadoc

Пример: Представление списка ресурсов в сервлете

Для представления списка ресурсов в сервлете применяется класс ResourceListRowData и один из классов HTMLFormConverter или HTMLTableConverter.

- Класс HTMLFormConverter показывает список ресурсов в виде последовательности форм, каждая из которых содержит значения атрибутов ресурса из списка.
- Класс HTMLTableConverter показывает список ресурсов в виде таблицы, строки которой содержат информацию о ресурсах из списка.

Колонки объекта ResourceListRowData задаются в виде массива ИД атрибутов колонки, а строки соответствуют объектам ресурсов.

```
   // Создание списка ресурсов. В этом примере создается
   // список всех сообщений в очереди сообщений
   // текущего пользователя.
   AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
   RMessageQueue messageQueue = new RMessageQueue(system, RMessageQueue.CURRENT);

   // Создать объект ResourceListRowData. Например, в таблице
// четыре колонки. В первой колонке находятся
// значки и имена всех сообщений из очереди.
   // В остальных колонках находится текст сообщения,
// серьезность и тип сообщения.
   ResourceListRowData rowdata = new ResourceListRowData(messageQueue,
       new Object[] { null, RQueuedMessage.MESSAGE_TEXT, RQueuedMessage.MESSAGE_SEVERITY,
                      RQueuedMessage.MESSAGE_TYPE } );

   // Создать объекты HTMLTable и HTMLTableConverter для
   // создания и настройки таблиц HTML.
   HTMLTable table = new HTMLTable();
   table.setCellSpacing(6);
   table.setBorderWidth(8);

   HTMLTableConverter converter = new HTMLTableConverter();
   converter.setTable(table);
   converter.setUseMetaData(true);

   // Создать таблицу HTML.
   String[] html = converter.convert(rowdata);
   System.out.println(html[0]);
```

Пример: Получение значения атрибута из класса Resource

One concrete subclass of the IBM Toolbox for Java Resource class is com.ibm.as400.resource.RUser, which represents a System i user. RUser supports many attribute IDs, each of which you can use to get attribute values.

В примере показано получение значения атрибута из класса RUser:

```
   // Создать объект RUser, соответствующий некоторому пользователю.
   AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
   RUser user = new RUser(system, "AUSERID");

   // Получить текстовое описание значения атрибута.
   String textDescription = (String)user.getAttributeValue(RUser.TEXT_DESCRIPTION);
```

Информация, связанная с данной

RUser Javadoc

Пример: Изменение значений атрибутов ресурса ChangeableResource

One concrete subclass of ChangeableResource is com.ibm.as400.resource.RJob, which represents a system job. RJob supports many attribute IDs, each of which you can use to access attribute values.

В примере показано изменение значений двух атрибутов RJob:

```
// Создать объект RJob, соответствующий конкретному заданию.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJob job = new RJob(system, "AJOBNAME", "AUSERID", "AJOBNUMBER");

// Задать значение для атрибута формата даты.
job.setAttributeValue(RJob.DATE_FORMAT, RJob.DATE_FORMAT_JULIAN);

// Задать значение для атрибута ИД страны или региона.
job.setAttributeValue(RJob.COUNTRY_ID, RJob.USER_PROFILE);

// Зафиксировать изменения значений атрибутов.
job.commitAttributeChanges();
```

RJob Javadoc

Пример: Доступ к ресурсам с помощью общего кода

Общий код позволяет выполнять операции с подклассами, дочерними по отношению к классам Resource, ResourceList и ChangeableResource. Такой код повышает возможности повторного применения и обслуживания программы и позволяет работать с новыми подклассами Resource, ResourceList и ChangeableResource без внесения изменений в программу.

Every attribute has an associated attribute meta data object (com.ibm.as400.resource.ResourceMetaData) that describes various properties of the attribute. Эти свойства включают возможные значения атрибута и значения по умолчанию, а также возможность изменения значения атрибута.

Examples

Пример: Печать содержимого объекта ResourceList

Ниже приведен пример общего кода, который отображает содержимое объекта ResourceList:

```
void printContents(ResourceList resourceList, long numberOfItems) throws ResourceException
{
    // Открыть список и подождать, пока не будет доступно
    // запрошенное число элементов.
    resourceList.open();
    resourceList.waitForResource(numberOfItems);

    for(long i = 0; i < numberOfItems; ++i)
    {
        System.out.println(resourceList.resourceAt(i));
    }
}
```

Пример: Обращение ко всем атрибутам ресурса с помощью класса ResourceMetaData

Ниже приведен пример общего кода, который отображает значения всех атрибутов ресурса:

```
void printAllAttributeValues(Resource resource) throws ResourceException
{
    // Получить мета-данные атрибута.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();

    // Перебрать все атрибуты и напечатать их значения.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        Object attributeID = attributeMetaData[i].getID();
```

```
        Object value = resource.getAttributeValue(attributeID);
        System.out.println("Атрибут " + attributeID + " = " + value);
    }
}
```

Пример: Сброс всех атрибутов класса ChangeableResource с помощью класса ResourceMetaData

Ниже приведен пример общего кода, который устанавливает значения по умолчанию всех атрибутов объекта ChangeableResource:

```
void resetAttributeValues(ChangeableResource resource) throws ResourceException
{
    // Получить мета-данные атрибута.
    ResourceMetaData[] attributeMetaData = resource.getAttributeMetaData();

    // Перебрать все атрибуты.
    for(int i = 0; i < attributeMetaData.length; ++i)
    {
        // Если значение атрибута может быть изменено, задать
        // значение по умолчанию.
        if (! attributeMetaData[i].isReadOnly())
        {
            Object attributeID = attributeMetaData[i].getID();
            Object defaultValue = attributeMetaData[i].getDefaultValue();
            resource.setAttributeValue(attributeID, defaultValue);
        }
    }

    // Зафиксировать все изменения значений атрибутов.
    resource.commitAttributeChanges();
}
```

Информация, связанная с данной

ResourceMetaData Javadoc

Примеры: RFML

This section lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java RFML component.

- Example: Using RFML compared to using IBM Toolbox for Java Record classes
- Пример: Исходный файл RFML

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Исходный файл RFML

This example RFML source file defines the format of customer records as used in the RFML example Using RFML compared to using IBM Toolbox for Java Record classes.

Исходным фалом RFML будет служить текстовый файл с именем qcustcdt.rfml.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
<?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>
```

Ссылки, связанные с данной

“Пример: Сравнение языка RFML с классами Record продукта IBM Toolbox for Java” на стр. 409
Этот пример демонстрирует различия в применении языка RFML и классов Record продукта IBM Toolbox for Java.

Пример: изменение владельца нити i5/OS с помощью разрешения

The following code example shows you how to use a profile token credential to swap the i5/OS thread identity and perform work on behalf of a specific user.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
// Prepare to work with the local system.
AS400 system = new AS400("localhost", "*CURRENT", "*CURRENT");

// Создание одноразового объекта ProfileTokenCredential на срок 60 секунд
// Должны быть заданы правильные ИД и пароль пользователя
ProfileTokenCredential pt = new ProfileTokenCredential();
pt.setSystem(system);
pt.setTimeoutInterval(60);
pt.setTokenType(ProfileTokenCredential.TYPE_SINGLE_USE);

pt.setTokenExtended("USERID", "PASSWORD");

// Смена владельца нити i5/OS с сохранением текущего разрешения
// для последующего восстановления исходной принадлежности нити
AS400Credential cr = pt.swap(true);

// Выполнение операций от имени нового владельца нити

// Восстановление исходной принадлежности нити i5/OS
cr.swap();

// Удаление разрешений
cr.destroy();
pt.destroy();
```

Примеры работы с классами сервлетов

The following examples show you some of the ways that you can use the servlet classes.

- Пример работы с классом ListRowData
- Пример работы с классом RecordListRowData
- Пример работы с классом SQLResultSetRowData
- Пример работы с классом HTMLFormConverter
- Пример работы с классом ListMetaData
- Пример работы с классом SQLResultSetMetaData
- Пример: Представление списка ресурсов сервлета

Классы сервлета могут применяться совместно с классами HTML, как показано в данном примере.

Отказ от гарантий на предоставляемый код

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.
>
> Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Работа с ListRowData

The three parts of this example illustrate using the ListRowData class to generate and display HTML.

Данный пример состоит из трех частей:

- “Исходный код на Java, демонстрирующий, как работает класс ListRowData”
- “Исходный код на HTML, созданный из исходного кода на Java с помощью HTMLTableConverter”
- “Представление кода HTML в браузере” на стр. 659

Исходный код на Java, демонстрирующий, как работает класс ListRowData

```
// Обращение к непустой очереди данных
KeyedDataQueue dq = new KeyedDataQueue(systemObject_, "/QSYS.LIB/MYLIB.LIB/MYDQ.DTAQ");

// Создание объекта метаданных.
ListMetaData metaData = new ListMetaData(2);

// Первый столбец будет содержать ИД заказчиков.
metaData.setColumnName(0, "ИД заказчика");
metaData.setColumnLabel(0, "ИД заказчика");
metaData.setColumnType(0, RowMetaDataType.STRING_DATA_TYPE);

// Второй столбец будет содержать номера заказов.
metaData.setColumnName(1, "Номер заказа");
metaData.setColumnLabel(1, "Номер заказа");
metaData.setColumnType(1, RowMetaDataType.STRING_DATA_TYPE);

// Создание объекта ListRowData.
ListRowData rowData = new ListRowData();
rowData.setMetaData(metaData);

// Получение записей из очереди данных
KeyedDataQueueEntry data = dq.read(key, 0, "EQ");
while (data != null)
{
   // Добавление записи очереди в таблицу
   Object[] row = new Object[2];
   row[0] = new String(key);
   row[1] = new String(data.getData());
   rowData.addRow(row);

   // Получение следующей записи из очереди.
   data = dq.read(key, 0, "EQ");
}

// Создание объекта преобразователя HTML и преобразование rowData в HTML.
HTMLTableConverter conv = new HTMLTableConverter();
conv.setUseMetaData(true);
HTMLTable[] html = conv.convertToTables(rowData);

// Просмотр результата преобразования.
System.out.println(html[0]);
```

Исходный код на HTML, созданный из исходного кода на Java с помощью HTMLTableConverter

С помощью класса “Класс HTMLTableConverter” на стр. 239 в примере на Java, приведенном выше, создается следующий код HTML.

```
<table>
<tr>
<th>Customer ID</th>
<th>Order Number</th>
</tr>
<tr>
```

```
<td>777-53-4444</td>
<td>12345-XYZ</td>
</tr>
<tr>
<td>777-53-4444</td>
<td>56789-ABC</td>
</tr>
</table>
```

Представление кода HTML в браузере

Ниже показано, как исходный код на HTML будет выглядеть в окне браузера.

ИД заказчика	Номер заказа
777-53-4444	12345-XYZ
777-53-4444	56789-ABC

Пример: Применение класса RecordListRowData

This example shows how the RecordListRowData class works, shows HTML source generated by using the HTMLTableConverted class, and also shows how a browser displays the generated HTML.

Пример состоит из трех частей:

- Java source that shows how the RecordListRowData class works
- Исходный код HTML, созданный кодом на Java с помощью класса HTMLTableConverter
- Представление кода HTML в браузере

Java source that shows how the RecordListRowData class works

```
// Создание объекта сервера.
AS400 mySystem = new AS400 ("mySystem.myComp.com", "UserId", "Password");

// Получение полного имени файла.
QSYSObjectPathName file = new QSYSObjectPathName(myLibrary, myFile, "%first%", "mbr");
String ifspath = file.getPath();

// Создание файлового объекта для представления файла.
SequentialFile sf = new SequentialFile(mySystem, ifspath);

// Получение формата записи из файла.
AS400FileRecordDescription recordDescription = new AS400FileRecordDescription(mySystem, ifspath);
RecordFormat recordFormat = recordDescription.retrieveRecordFormat()[0];

// Задание формата записи для файла.
sf.setRecordFormat(recordFormat);

// Считывание записей из файла.
Record[] records = sf.readAll();

// Создание объекта RecordListRowData и добавление записей.
RecordListRowData rowData = new RecordListRowData(recordFormat);

for (int i=0; i < records.length; i++)
{
   rowData.addRow(records[i]);
}

// Создание объекта преобразователя HTML и преобразование rowData в HTML.
HTMLTableConverter conv = new HTMLTableConverter();
conv.setMaximumTableSize(3);
```

```
HTMLTable[] html = conv.convertToTables(rowData);

// Просмотр первой сформированной преобразователем таблицы HTML.
System.out.println(html[0]);
```

Исходный код HTML, созданный исходным кодом на Java с помощью класса HTMLTableConverter

Класс HTMLTableConverter в приведенном выше примере программы на Java создает следующий текст в формате HTML.

```
<table>
<tr>
<th>CNUM</th>
<th>LNAM</th>
<th>INIT</th>
<th>STR</th>
<th>CTY</th>
<th>STATE</th>
<th>ZIP</th>
<th>CTLMT</th>
<th>CHGCOD</th>
<th>BDUE</th>
<th>CTDUE</th>
</tr>
<tr>
<td>938472</td>
<td>Henning </td>
<td>G K</td>
<td>4859 Elm Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75217</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">37.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>839283</td>
<td>Jones   </td>
<td>B D</td>
<td>21B NW 135 St</td>
<td>Clay  </td>
<td>NY</td>
<td align="right">13041</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">100.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>392859</td>
<td>Vine    </td>
<td>S S</td>
<td>PO Box 79    </td>
<td>Broton</td>
<td>VT</td>
<td align="right">5046</td>
<td align="right">700</td>
<td align="right">1</td>
<td align="right">439.00</td>
<td align="right">0.00</td>
</tr>
</table>
```

Представления кода HTML в браузере

В приведенной ниже таблице показано, как исходный код на HTML будет выглядеть в окне браузера.

CNUM	LNAM	INIT	STR	CTY	STATE	ZIP	CTLMT	CHGCOD	BDUE	CTDUE
938472	Henning	G K	4859 Elm Ave	Dallas	TX	75217	5000	3	37.00	0.00
839283	Jones	B D	21B NW 135 St	Clay	NY	13041	400	1	100.00	0.00
392859	Vine	S S	PO Box 79	Broton	VT	5046	700	1	439.00	0.00

Пример: Применение класса SQLResultSetRowData

This example shows how the SQLResultSetRowData class works, shows the generated HTML source, and also shows how a browser displays the generated HTML.

Данный пример состоит из трех частей:

- Java source that shows how the SQLResultSetRowData class works
- Исходный код HTML, созданный кодом на Java с помощью класса HTMLTableConverter
- Представление кода HTML в браузере

Java source that shows how the SQLResultSetRowData class works

```
// Создание объекта сервера.
AS400 mySystem = new AS400 ("mySystem.myComp.com", "ИД_пользователя", "Пароль");

// Регистрация и подключение к базе данных.
DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
Connection connection = DriverManager.getConnection("jdbc:as400://" + mySystem.getSystemName());

// Выполнение оператора SQL и получение набора результатов.
Statement statement = connection.createStatement();
statement.execute("select * from qiws.qcustcdt");
ResultSet resultSet = statement.getResultSet();

// Создание объекта SQLResultSetRowData для инициализации набора результатов.
SQLResultSetRowData rowData = new SQLResultSetRowData(resultSet);

// Создание объекта таблицы HTML, который будет применяться в ходе преобразования.
HTMLTable table = new HTMLTable();

// Определение заголовков столбцов.
String[] headers = {"Customer Number", "Last Name", "Initials",
                                       "Street Address", "City", "State", "Zip Code",
                                       "Credit Limit", "Charge Code", "Balance Due",
                                       "Credit Due"};
table.setHeader(headers);

// Установка опций форматирования таблицы.
table.setBorderWidth(2);
table.setCellSpacing(1);
table.setCellPadding(1);

// Создание объекта преобразователя HTML и преобразование rowData в HTML.
HTMLTableConverter conv = new HTMLTableConverter();
conv.setTable(table);
HTMLTable[] html = conv.convertToTables(rowData);

// Вывод первой сформированной преобразователем таблицы HTML.
System.out.println(html[0]);
```

Исходный код HTML, созданный исходным кодом на Java с помощью класса HTMLTableConverter

Класс HTMLTableConverter в приведенном выше примере программы на Java создает следующий текст в формате HTML.

```
<table border="2" cellpadding="1" cellspacing="1">
<tr>
<th>Customer Number</th>
<th>Last Name</th>
<th>Initials</th>
<th>Street Address</th>
<th>City</th>
<th>State</th>
<th>Zip Code</th>
<th>Credit Limit</th>
<th>Charge Code</th>
<th>Balance Due</th>
<th>Credit Due</th>
</tr>
<tr>
<td>938472</td>
<td>Henning </td>
<td>G K</td>
<td>4859 Elm Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75217</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">37.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>839283</td>
<td>Jones   </td>
<td
>B D</td>
<td>21B NW 135 St</td>
<td>Clay  </td>
<td>NY</td>
<td align="right">13041</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">100.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>392859</td>
<td>Vine    </td>
<td>S S</td>
<td>PO Box 79    </td>
<td>Broton</td>
<td>VT</td>
<td align="right">5046</td>
<td align="right">700</td>
<td align="right">1</td>
<td align="right">439.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>938485</td>
<td>Johnson </td>
<td>J A</td>
<td>3 Alpine Way </td>
<td>Helen </td>
<td>GA</td>
<td align="right">30545</td>
<td align="right">9999</td>
<td align="right">2</td>
<td align="right">3987.50</td>
<td align="right">33.50</td>
</tr>
```

```
<tr>
<td>397267</td>
<td>Tyron   </td>
<td>W E</td>
<td>13 Myrtle Dr </td>
<td>Hector</td>
<td>NY</td>
<td align="right">14841</td>
<td align="right">1000</td>
<td align="right">1</td>
<td align="right">0.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>389572</td>
<td>Stevens </td>
<td>K L</td>
<td>208 Snow Pass</td>
<td>Denver</td>
<td>CO</td>
<td align="right">80226</td>
<td align="right">400</td>
<td align="right">1</td>
<td align="right">58.75</td>
<td align="right">1.50</td>
</tr>
<tr>
<td>846283</td>
<td>Alison  </td>
<td>J S</td>
<td>787 Lake Dr  </td>
<td>Isle  </td>
<td>MN</td>
<td align="right">56342</td>
<td align="right">5000</td>
<td align="right">3</td>
<td align="right">10.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>475938</td>
<td>Doe     </td>
<td>J W</td>
<td>59 Archer Rd </td>
<td>Sutter</td>
<td>CA</td>
<td align="right">95685</td>
<td align="right">700</td>
<td align="right">2</td>
<td align="right">250.00</td>
<td align="right">100.00</td>
</tr>
<tr>
<td>693829</td>
<td>Thomas  </td>
<td>A N</td>
<td>3 Dove Circle</td>
<td>Casper</td>
<td>WY</td>
<td align="right">82609</td>
<td align="right">9999</td>
<td align="right">2</td>
<td align="right">0.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>593029</td>
```

```
<td>Williams</td>
<td>E D</td>
<td>485 SE 2 Ave </td>
<td>Dallas</td>
<td>TX</td>
<td align="right">75218</td>
<td align="right">200</td>
<td align="right">1</td>
<td align="right">25.00</td>
<td align="right">0.00</td>
</tr>
<tr>
<td>192837</td>
<td>Lee     </td>
<td>F L</td>
<td>5963 Oak St  </td>
<td>Hector</td>
<td>NY</td>
<td align="right">14841</td>
<td align="right">700</td>
<td align="right">2</td>
<td align="right">489.50</td>
<td align="right">0.50</td>
</tr>
<tr>
<td>583990</td>
<td>Abraham </td>
<td>M T</td>
<td>392 Mill St  </td>
<td>Isle  </td>
<td>MN</td>
<td align="right">56342</td>
<td align="right">9999</td>
<td align="right">3</td>
<td align="right">500.00</td>
<td align="right">0.00</td>
</tr>
</table>
```

Представление кода HTML в браузере

В приведенной ниже таблице показано, как исходный код на HTML будет выглядеть в окне браузера.

Customer Number	Last Name	Initials	Street Address	City	State	Zip Code	Credit Limit	Charge Code	Balance Due	Credit Due
938472	Henning	G K	4859 Elm Ave	Dallas	TX	75217	5000	3	37.00	0.00
839283	Jones	B D	21B NW 135 St	Clay	NY	13041	400	1	100.00	0.00
392859	Vine	S S	PO Box 79	Broton	VT	5046	700	1	439.00	0.00
938485	Johnson	J A	3 Alpine Way	Helen	GA	30545	9999	2	3987.50	33.50
397267	Tyron	W E	13 Myrtle Dr	Hector	NY	14841	1000	1	0.00	0.00
389572	Stevens	K L	208 Snow Pass	Denver	CO	80226	400	1	58.75	1.50
846283	Alison	J S	787 Lake Dr	Isle	MN	56342	5000	3	10.00	0.00
475938	Doe	J W	59 Archer Rd	Sutter	CA	95685	700	2	250.00	100.00

Customer Number	Last Name	Initials	Street Address	City	State	Zip Code	Credit Limit	Charge Code	Balance Due	Credit Due
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

Пример: Применение класса HTMLFormConverter

While running a web server with servlet support, compile and run the following example to see how the HTMLFormConverter works.

```
import java.awt.Color;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.util.html.GridLayoutFormPanel;
import com.ibm.as400.util.html.HTMLConstants;
import com.ibm.as400.util.html.HTMLForm;
import com.ibm.as400.util.html.HTMLTable;
import com.ibm.as400.util.html.HTMLTableCaption;
import com.ibm.as400.util.html.HTMLText;
import com.ibm.as400.util.html.LabelFormElement;
import com.ibm.as400.util.html.LineLayoutFormPanel;
import com.ibm.as400.util.html.SubmitFormInput;
import com.ibm.as400.util.html.TextFormInput;

import com.ibm.as400.util.servlet.HTMLFormConverter;
import com.ibm.as400.util.servlet.SQLResultSetRowData;

import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400JDBCDriver;

/**
 * Пример использования класса HTMLFormConverter в сервлете.
 */
public class HTMLFormConverterExample extends HttpServlet

{
   private String userId_   = "myUserId";
   private String password_ = "myPwd";
   private AS400 system_;

   private Connection databaseConnection_;



   // Очистка перед возвратом к основной форме HTML.
   public void cleanup()
   {
      try
      {
         // Завершение соединения с базой данных.
```

```
      if (databaseConnection_ != null)
      {
         databaseConnection_.close();
         databaseConnection_ = null;
      }
   }
   catch (Exception e)
   {
      e.printStackTrace ();
   }
}


// Преобразование набора строк в текст формата HTML.
private HTMLTable[] convertRowData(SQLResultSetRowData rowData)
{
   try
   {
      // Создание объекта преобразования, который сформирует HTML
      // на основе результата запроса, переданного базе данных.
               HTMLFormConverter converter = new HTMLFormConverter();

      // Установка атрибутов формы.
      converter.setBorderWidth(3);
      converter.setCellPadding(2);
      converter.setCellSpacing(4);

      // Преобразование набора записей в форматированный текст HTML.
      HTMLTable[] htmlTable = converter.convertToForms(rowData);
      return htmlTable;
   }
   catch (Exception e)
   {
      e.printStackTrace ();
      return null;
   }

}


// Возврат ответа клиенту.
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws IOException, ServletException
{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println(showHtmlMain());
   out.close();
}


// Обработка данных, введенных в форме.
public void doPost (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
{
   SQLResultSetRowData rowData = new SQLResultSetRowData();
   HTMLTable[] htmlTable = null;

   // Получение текущего объекта сеанса или создание нового.
   HttpSession session = request.getSession(true);

   ServletOutputStream out = response.getOutputStream();

   response.setContentType("text/html");

   Hashtable parameters = getRequestParameters (request);
```

```
// Получение записей и значений таблицы HTML для этого сеанса.
rowData = (SQLResultSetRowData) session.getValue("sessionRowData");
htmlTable = (HTMLTable[]) session.getValue("sessionHtmlTable");

// Если это первый проход - вывод первой записи
if (parameters.containsKey("getRecords"))
{
   rowData = getAllRecords(parameters, out);

   if (rowData != null)
   {
      // Установка данных из записи для этого сеанса.
      session.putValue("sessionRowData", rowData);

      // Переход к первой записи.
      rowData.first();

      // Преобразование набора строк в текст формата HTML.
      htmlTable = convertRowData(rowData);

      if (htmlTable != null)
      {
         rowData.first();
         session.putValue("sessionHtmlTable", htmlTable);
         out.println(showHtmlForRecord(htmlTable, 0));
      }
   }
}
// Если будет нажата кнопка Возврат в главную форму,
// пользователь перейдет к главной форме HTML
else if (parameters.containsKey("returnToMain"))
{
   session.invalidate();
   cleanup();
   out.println(showHtmlMain());
}
// При нажатии кнопки Первая будет показана первая запись
else if (parameters.containsKey("getFirstRecord"))
{
   rowData.first();
   out.println(showHtmlForRecord(htmlTable, 0));
}
// При нажатии кнопки Предыдущая будет показана предыдущая запись
else if (parameters.containsKey("getPreviousRecord"))
{
   if (!rowData.previous())
   {
      rowData.first();
   }
   out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
}
// При нажатии кнопки Следующая будет показана следующая запись
else if (parameters.containsKey("getNextRecord"))
{
   if (!rowData.next())
   {
      rowData.last();
   }
   out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
}
// При нажатии кнопки Последняя будет показана последняя запись
else if (parameters.containsKey("getLastRecord"))
{
   rowData.last();
   out.println(showHtmlForRecord(htmlTable, rowData.getCurrentPosition()));
}
```

```
      // Если ни одно из предыдущих условий не выполнено - возможно, произошла ошибка.
      else
      {
         out.println(showHtmlForError("Произошла внутренняя ошибка.
Непредвиденные значения параметров."));
      }

      // Сохранение данных из записей для текущего сеанса для обновления
      // текущей позиции в объекте, связанном с этим сеансом.
      session.putValue("sessionRowData", rowData);

      // Закрытие потока вывода.
      out.close();
   }


   // Выборка всех записей из файла, указанного пользователем.
   private SQLResultSetRowData getAllRecords(Hashtable parameters, ServletOutputStream out)
      throws IOException
   {
      SQLResultSetRowData records = null;

      try
      {
         // Выборка имен системы, библиотеки и файла из списка параметров.
         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("Неверное имя системы, файла или библиотеки."));
         }
         else
         {
            // Получение соединения с сервером.
            getDatabaseConnection (sys, out);
            if (databaseConnection_ != null)
            {
               Statement sqlStatement = databaseConnection_.createStatement();

               // Формирование запроса для выборки записей.
               String query = "SELECT * FROM " + lib + "." + file;
               ResultSet rs = sqlStatement.executeQuery (query);

               boolean rsHasRows = rs.next();  // поместить курсор на первую запись.

               // При отсутствии записей вывод сообщения об ошибке,
               // в противном случае - сохранение выбранных данных.
               if (!rsHasRows)
               {
                  out.println(showHtmlForError("В файле нет записей."));
               }
               else
               {
                  records = new SQLResultSetRowData (rs);
               }

               // Не закрывайте оператор с помощью метода ResultSet до завершения операции; в противном
               // случае может произойти сбой.
               sqlStatement.close();
            }
         }
      }
      catch (Exception e)
      {
```

```
      e.printStackTrace ();
      out.println(showHtmlForError(e.toString()));
   }

   return records;
}


// Установление соединения с базой данных.
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()));
      }
   }
}


// Получение параметров запроса к сервлету 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;
}


// Получение информации о сервлете.
public String getServletInfo()
{
   return "HTMLFormConverterExample";
}


// Инициализация.
public void init(ServletConfig config)
{
   try
   {
      super.init(config);

      // Регистрация драйвера JDBC.
      try
      {
         DriverManager.registerDriver(new AS400JDBCDriver());
      }
      catch (Exception e)
      {
         System.out.println("Драйвер JDBC не найден");
      }
   }
```

```
   catch (Exception e)
   {
      e.printStackTrace();
   }
}


// Установка параметров заголовка страницы.
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 ();
}


// Вывод страницы HTML с информацией об ошибке.
private String showHtmlForError(String message)
{
   String title = "Ошибка";
   StringBuffer page = new StringBuffer();
   page.append (showHeader (title));
   try
   {
      // Создание объекта формы HTML.
      HTMLForm errorForm = new HTMLForm("HTMLFormConverterExample");

      // Настройка вызова doPost() при передаче формы на обработку.
               errorForm.setMethod(HTMLForm.METHOD_POST);

      // Создание панели с одним столбцом, в который будут
      // добавлены элементы HTML.
      GridLayoutFormPanel grid = new GridLayoutFormPanel();

      // Создание текстового элемента для сообщений об ошибках и добавление его в панель.
      HTMLText text = new HTMLText(message);
      text.setBold(true);
      text.setColor(Color.red);
      grid.addElement(text);

      // Создание кнопки для возврата к главной странице и добавление ее в панель.
      grid.addElement(new SubmitFormInput("returnToMain", "Return to Main"));

      // Добавление панели в форму HTML.
      errorForm.addElement(grid);

      page.append(errorForm.toString());
   }
   catch (Exception e)
   {
      e.printStackTrace ();
   }
   page.append("</body></html>");
   return page.toString();
}


// Вывод формы HTML для отдельной записи.
private String showHtmlForRecord(HTMLTable[] htmlTable, int position)
{
   String title = "HTMLFormConverter Example";
   StringBuffer page = new StringBuffer();
   page.append (showHeader (title));

   page.append("<h1>" + title + "</h1>");
```

```
   try
   {
      // Создание объекта формы HTML.
      HTMLForm recForm = new HTMLForm("HTMLFormConverterExample");

      // Настройка вызова doPost() при передаче формы на обработку.
               recForm.setMethod(HTMLForm.METHOD_POST);

      // Создание одной панели, на которой будут расположены
      // созданные элементы HTML.
      GridLayoutFormPanel grid = new GridLayoutFormPanel();

      // Создание и добавление названия таблицы, в которую
      // будут заноситься данные для текущей записи.
      HTMLText recNumText = new HTMLText("Номер записи: " + (position + 1));
      recNumText.setBold(true);
      grid.addElement(recNumText);

      // Создание двух панелей, на которых будут расположены
      // таблица и текст комментария.
      GridLayoutFormPanel tableGrid = new GridLayoutFormPanel(2);
      tableGrid.addElement(htmlTable[position]);
      HTMLText comment = new HTMLText("   <---- Вывод класса HTMLFormConverter");
      comment.setBold(true);
      comment.setColor(Color.blue);
      tableGrid.addElement(comment);

      // Добавление строки таблицы в панель.
      grid.addElement(tableGrid);

      // Создание одной панели, на которой будут расположены
      // кнопки перемещения по набору записей.
      LineLayoutFormPanel buttonLine = new LineLayoutFormPanel();
      buttonLine.addElement(new SubmitFormInput("getFirstRecord", "Первая"));
      buttonLine.addElement(new SubmitFormInput("getPreviousRecord", "Предыдущая"));
      buttonLine.addElement(new SubmitFormInput("getNextRecord", "Следующая"));
      buttonLine.addElement(new SubmitFormInput("getLastRecord", "Последняя"));

      // Настройка еще одной разметки панели с одной строкой для кнопки
      // Возврат к основной форме.
      LineLayoutFormPanel returnToMainLine = new LineLayoutFormPanel();
      returnToMainLine.addElement(new SubmitFormInput("returnToMain", "Return to Main"));

      // Добавление строк с кнопками в панель с таблицей.
      grid.addElement(buttonLine);
      grid.addElement(returnToMainLine);

      // Добавление панели в форму.
      recForm.addElement(grid);

      // Добавление формы в страницу HTML.
      page.append(recForm.toString());
   }
   catch (Exception e)
   {
      e.printStackTrace ();
   }

   page.append("</body></html>");
   return page.toString();
}


// Показать основную форму HTML (запрос имен системы, файла и
// библиотеки).
private String showHtmlMain()
{
```

```
String title = "HTMLFormConverter Example";
StringBuffer page = new StringBuffer();
page.append (showHeader (title));

page.append("<h1>" + title + "</h1>");

// Создание объекта формы HTML.
HTMLForm mainForm = new HTMLForm("HTMLFormConverterExample");

try
{
   // Настройка вызова doPost() при передаче формы на обработку.
            mainForm.setMethod(HTMLForm.METHOD_POST);

   // Добавление краткого описания формы.
   HTMLText desc =
      new HTMLText("<P>This example uses the HTMLFormConverter class " +
                   "для преобразования данных, полученных из файла" +
                   "сервера.  При преобразовании создается массив таблиц HTML. " +
                   "  Каждая запись в массиве - это запись файла." +
                   "  " +
                   "Записи выводятся по одной с " +
                   "кнопками перехода вперед и назад по списку " +
                   "through the list of records.</P>");
   mainForm.addElement(desc);

   // Добавление инструкций в форму.
   HTMLText instr =
      new HTMLText("<P>Please input the name of the server, " +
                   "имена файла и библиотеки, которые нужно " +
                   "считать.  После этого нажмите кнопку Показать записи " +
                   "button to continue.</P>");
   mainForm.addElement(instr);


   // Создание панели макета сетки и добавление полей ввода
   // имен системы, файла и библиотеки.
   GridLayoutFormPanel panel = new GridLayoutFormPanel(2);

   LabelFormElement sysPrompt = new LabelFormElement("Сервер: ");
   TextFormInput system = new TextFormInput("System");
   system.setSize(10);

   LabelFormElement filePrompt = new LabelFormElement("Имя файла: ");
   TextFormInput file = new TextFormInput("File");
   file.setSize(10);

   LabelFormElement libPrompt = new LabelFormElement("Имя библиотеки: ");
   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);

   // Добавление панели в форму.
   mainForm.addElement(panel);

   // Создание кнопки обработки формы и добавление ее в форму.
   mainForm.addElement(new SubmitFormInput("getRecords", "Show Records"));
}
catch (Exception e)
{
   e.printStackTrace ();
```

```
      }

      page.append(mainForm.toString());
      page.append("</body></html>");

      return page.toString();
   }

}
```

В результате будет создан следующий код HTML:

```
<table border="0">
<tr>
<td><b>Record number: 1</b></td>
</tr>
<tr>
<td><table border="0">
<tr>
<td><table border="3" cellpadding="2" cellspacing="4">
<tr>
<th>CUSNUM</th>
<td>839283</td>
</tr>
<tr>
<th>LSTNAM</th>
<td>Jones   </td>
</tr>
<tr>
<th>INIT</th>
<td>B D</td>
</tr>
<tr>
<th>STREET</th>
<td>21B NW 135 St</td>
</tr>
<tr>
<th>CITY</th>
<td>Clay  </td>
</tr>
<tr>
<th>STATE</th>
<td>NY</td>
</tr>
<tr>
<th>ZIPCOD</th>
<td>13041</td>
</tr>
<tr>
<th>CDTLMT</th>
<td>400</td>
</tr>
<tr>
<th>CHGCOD</th>
<td>1</td>
</tr>
<tr>
<th>BALDUE</th>
<td>100.00</td>
</tr>
<tr>
<th>CDTDUE</th>
<td>0.00</td>
</tr>
</table>
</td>
<td><font color="#0000ff"> <b><!-- Output from the HTMLFormConverter class-->
```

```
</b></font></td>
</tr>
</table>
</td>
</tr>
<tr>
<form>
<td><input type="submit" name="getFirstRecord" value="First" />
<input type="submit" name="getPreviousRecord" value="Previous" />
<input type="submit" name="getNextRecord" value="Next" />
<input type="submit" name="getLastRecord" value="Last" />
<br />
</td>
</tr>
<tr>
<td><input type="submit" name="returnToMain" value="Return to Main" />
<br />
</td>
</tr>
</table>
</form>
```

Пример работы с классами HTML и классами сервлетов

Ниже приведет пример работы с классами HTML и классами сервлетов. Он дает общее представление об их применении. Откомпилируйте и запустите этот пример, предварительно убедившись, что запущены Web-сервер и браузер.

```
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.*;

/*
Пример применения классов IBM Toolbox for Java в сервлете.

Схемы базы данных SQL на сервере:

File . . . . . .   LICENSES
Library  . . .     LIGHTSON

 Field            Type          Length   Nulls
 LICENSE          CHARACTER         10   NOT NULL
 USER_ID          CHARACTER         10   NOT NULL WITH DEFAULT
 E_MAIL           CHARACTER         20   NOT NULL
 WHEN_ADDED       DATE                   NOT NULL WITH DEFAULT
 TIME_STAMP       TIMESTAMP              NOT NULL WITH DEFAULT


File . . . . . .   REPORTS
Library  . . .     LIGHTSON

 Field            Type          Length   Nulls
 LICENSE          CHARACTER         10   NOT NULL
 REPORTER         CHARACTER         10   NOT NULL WITH DEFAULT
 DATE_ADDED       DATE                   NOT NULL WITH DEFAULT
 TIME_ADDED       TIME                   NOT NULL WITH DEFAULT
 TIME_STAMP       TIMESTAMP              NOT NULL WITH DEFAULT
 LOCATION         CHARACTER         10   NOT NULL
 COLOR            CHARACTER         10   NOT NULL
```

```
 CATEGORY         CHARACTER         10   NOT NULL
*/

public class LightsOn extends javax.servlet.http.HttpServlet

{
  private AS400 system_;
  private String password_;  // пароль для сервера и базы данных 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 { // Ни одно из перечисленных выше условий не выполнено.
           // Предполагается, что пользователь заполнил форму
           // и передал ее на обработку. Программа перехватывает
// поступающую информацию и выполняет запрошенное действие.

      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("Ошибка (внутренняя): " +
                        "Отлично от Report, Register, " +
                        "Unregister, ListRegistered или ListReported."));
    }
  }

  out.close(); // Закрытие потока вывода.

}

// Считывание параметров из запроса к сервлету HTTP и
// сохранение их в хэш-таблице.
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;
}


// Удаление из строки пробелов и дефисов и преобразование всех букв в прописные.
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();
}

// Создание списка строк, заключенных в одинарные кавычки.
private static String quoteList (String[] inList)
{
  StringBuffer outList = new StringBuffer();
  for (int i=0; i<inList.length; i++)
  {
    outList.append ("'" + inList[i] + "'");
    if (i<inList.length-1)
      outList.append (",");
  }
  return outList.toString();
}


public String getServletInfo ()
{
  return "Lights-On Servlet";
```

```
}

private AS400 getSystem ()
{
  try
  {
    if (system_ == null)
    {
      system_ = new AS400();

      // Примечание: Рекомендуется брать значения из
      // файла свойств.
      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);

    // Регистрация драйвера JDBC.
    try {
      java.sql.DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());
    }
    catch (Exception e)
    {
      System.out.println("Драйвер JDBC не найден");
    }


  }
  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 ("Ошибка: Не указан номерной знак.\n");

  if (eMailAddress == null || eMailAddress.length() == 0)
    acknowledgement.append ("Ошибка: Не указан электронный адрес для уведомления.\n");

  if (acknowledgement.length() == 0)
  {
    try
    {
      // Запись нового номерного знака и электронного адреса в базу данных.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Отправка запроса.
      String cmd = "INSERT INTO LICENSES (LICENSE, E_MAIL) VALUES (" +
        quoteList(new String[] {licenseNum, eMailAddress}) + ")";
      sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

      // Подтверждение получения запроса.
      acknowledgement.append ("Номерной знак " + licenseNum + " зарегистрирован.");
      acknowledgement.append ("Электронный адрес для уведомления: " + 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 ("Ошибка: Не указан номерной знак.\n");

  if (acknowledgement.length() == 0)
  {
    try
    {
      // Удаление заданного номерного знака и электронного адреса из базы данных.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Удаление строки из базы данных LICENSES.
      String cmd = "DELETE FROM LICENSES WHERE LICENSE = '" + licenseNum + "'";
      sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

      // Подтверждение получения запроса.
      acknowledgement.append ("Номерной знак " + licenseNum + " удален из базы данных.");
    }
    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 ("Ошибка: Не указан номерной знак.");

  if (acknowledgement.length() == 0)
  {
    try
    {
      // Передача информации об указанном автомобиле.
      getDatabaseConnection ();
      Statement sqlStatement = databaseConnection_.createStatement();

      // Добавление записи в базу данных REPORTS.
      String cmd = "INSERT INTO REPORTS (LICENSE, LOCATION, COLOR, CATEGORY) VALUES (" +
        quoteList(new String[] {licenseNum, location, color, category}) + ")";
      sqlStatement.executeUpdate(cmd);
      sqlStatement.close();

      // Подтверждение получения запроса.
      acknowledgement.append ("Получены данные о лицензии с номером " +
      licenseNum + ".  Благодарим вас!");
    }
    catch (Exception e) { e.printStackTrace (); }
  }
  return acknowledgement.toString();
}


private String showHeader (String title)
{
  StringBuffer page = new StringBuffer();
  page.append("<html><head><title>" + title + "</title>");
  page.append("</head><body bgcolor=\"blanchedalmond\">");
  return page.toString ();
}


private String showAcknowledgement (String acknowledgement)
{
  String title = "Acknowledgement";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  try {
    HTMLForm form = new HTMLForm("LightsOn");
    GridLayoutFormPanel grid = new GridLayoutFormPanel();
    HTMLText text = new HTMLText(acknowledgement);
    if (acknowledgement.startsWith("Error"))
      text.setBold(true);
    grid.addElement(text);
    grid.addElement(new SubmitFormInput("returningToMain", "В начало"));
    form.addElement(grid);
    page.append(form.toString());
  }
  catch (Exception e) { e.printStackTrace (); }
  page.append("</body></html>");
  return page.toString();
}


private String showHtmlMain ()
{
```

```
  String title = "Средство Lights-On (сведения)";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");


  // Создание объекта формы HTML.
  HTMLForm mainForm = new HTMLForm("LightsOn");
  GridLayoutFormPanel grid = new GridLayoutFormPanel();

  try {
    // Настройка вызова doPost() при передаче формы на обработку.
    mainForm.setMethod(HTMLForm.METHOD_POST);

    // Создание кнопок.
    grid.addElement(new SubmitFormInput("askingToReport",
                    "Сообщить сведения об автомобиле"));
    grid.addElement(new SubmitFormInput("askingToRegister",
                    "Зарегистрировать номерной знак"));
    grid.addElement(new SubmitFormInput("askingToUnregister",
                    "Удалить номерной знак из базы данных"));
    grid.addElement(new SubmitFormInput("askingToListRegistered",
                    "Показать все зарегистрированные номера"));
    grid.addElement(new SubmitFormInput("askingToListReported",
                    "Показать все автомобили, о которых есть сведения"));

    mainForm.addElement(grid);
  }
  catch (Exception e) { e.printStackTrace (); }

  page.append(mainForm.toString());
  page.append("</body></html>");

  return page.toString();
}


private String showHtmlForReporting ()
{
  String title = "Сообщить сведения об автомобиле";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

  // Создание объекта формы HTML.
  HTMLForm reportForm = new HTMLForm("LightsOn");
  GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

  try {
    // Настройка вызова doPost() при передаче формы на обработку.
    reportForm.setMethod(HTMLForm.METHOD_POST);

    TextFormInput licenseNum = new TextFormInput("licenseNum");
    licenseNum.setSize(10);
    licenseNum.setMaxLength(10);

    // Добавление элементов в линейную форму.
    grid.addElement(new LabelFormElement("Номерной знак автомобиля:"));
    grid.addElement(licenseNum);

    // Создание группы радиокнопок и самих радиокнопок.
          RadioFormInputGroup colorGroup = new RadioFormInputGroup("color");

    colorGroup.add("color", "white", "white", true);
    colorGroup.add("color", "black", "black", false);
```

```
    colorGroup.add("color", "gray", "gray", false);
    colorGroup.add("color", "red", "red", false);
    colorGroup.add("color", "yellow", "yellow", false);
    colorGroup.add("color", "green", "green", false);
    colorGroup.add("color", "blue", "blue", false);
    colorGroup.add("color", "brown", "brown", false);

    // Создание списка классов автомобилей.
    SelectFormElement category = new SelectFormElement("category");
    category.addOption("sedan", "sedan", true);
    category.addOption("convertible", "convertibl"); // Поле БД длиной 10 символов
    category.addOption("truck", "truck");
    category.addOption("van", "van");
    category.addOption("SUV", "SUV");
    category.addOption("motorcycle", "motorcycle");
    category.addOption("other", "other");

    // Создание списка расположений автомобилей (номеров домов).
    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("Цвет:"));
    grid.addElement(colorGroup);

    grid.addElement(new LabelFormElement("Класс автомобиля:"));
    grid.addElement(category);

    grid.addElement(new LabelFormElement("Дом:"));
    grid.addElement(location);

    grid.addElement(new SubmitFormInput("submittingReport", "Отправить отчет"));
    grid.addElement(new SubmitFormInput("returningToMain", "В начало"));

    reportForm.addElement(grid);
  }
  catch (Exception e) { e.printStackTrace (); }

  page.append(reportForm.toString());

  page.append("</body></html>");

  return page.toString();
}


private String showHtmlForRegistering ()
{
  String title = "Зарегистрировать номерной знак";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

  // Создание объекта формы HTML.
  HTMLForm registrationForm = new HTMLForm("LightsOn");

  // Создание двух панелей, на которых будут расположены
  // созданные элементы HTML.
  GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

  try {
    // Настройка вызова doPost() при передаче формы на обработку.
```

```
    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("Номерной знак:"));
    grid.addElement(licenseNum);

    grid.addElement(new LabelFormElement("Электронный адрес для уведомления:"));
    grid.addElement(eMailAddress);

    grid.addElement(new SubmitFormInput("submittingRegistration", "Зарегистрировать"));
    grid.addElement(new SubmitFormInput("returningToMain", "В начало"));

    registrationForm.addElement(grid);
  }
  catch (Exception e) { e.printStackTrace (); }

  page.append(registrationForm.toString());

  page.append("</body></html>");

  return page.toString();
}


private String showHtmlForUnregistering ()
{
  String title = "Удалить номерной знак из базы данных";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  page.append("<h1>" + title + "</h1>");

  // Создание объекта формы HTML.
  HTMLForm unregistrationForm = new HTMLForm("LightsOn");
  GridLayoutFormPanel grid = new GridLayoutFormPanel(2);

  try {
    // Настройка вызова doPost() при передаче формы на обработку.
    unregistrationForm.setMethod(HTMLForm.METHOD_POST);

    // Создание объекта LineLayoutFormPanel.
    TextFormInput licenseNum = new TextFormInput("licenseNum");
    licenseNum.setSize(10);
    licenseNum.setMaxLength(10);

    grid.addElement(new LabelFormElement("Номерной знак автомобиля:"));
    grid.addElement(licenseNum);

    grid.addElement(new SubmitFormInput("submittingUnregistration", "Удалить из базы данных"));
    grid.addElement(new SubmitFormInput("returningToMain", "В начало"));

    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 = "Все зарегистрированные номерные знаки";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  try
  {
    // Создание объекта формы HTML.
    HTMLForm mainForm = new HTMLForm("LightsOn");

    // Создание одной панели, на которой будут расположены
    // созданные элементы HTML.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

    // Выбор расположения элементов созданной таблицы.
    HTMLTable table = new HTMLTable();
    table.setAlignment(HTMLConstants.LEFT);
    table.setBorderWidth(3);

    // Создание и добавление названия и заголовка таблицы.
    HTMLTableCaption caption = new HTMLTableCaption();
    caption.setAlignment(HTMLConstants.TOP);
    caption.setElement(title);
    table.setCaption(caption);
    table.setHeader(new String[] { "Знак", "Дата добавления" } );

    // Создание программы преобразования, формирующей таблицу HTML из таблицы
    // результатов, полученной в результате обработки запроса к базе данных.
    HTMLTableConverter converter = new HTMLTableConverter();
    converter.setTable(table);

    getDatabaseConnection ();
    Statement sqlStatement = databaseConnection_.createStatement();

    // Проверка, не пуста ли база данных.
    String query = "SELECT COUNT(*) FROM LICENSES";
    ResultSet rs = sqlStatement.executeQuery (query);
    rs.next();  // поместить курсор в первую строку
    int rowCount = rs.getInt(1);

    if (rowCount == 0) {
      page.append ("<font size=4 color=red>No vehicles have been reported.</font>");
    }
    else {
      query = "SELECT LICENSE,WHEN_ADDED FROM LICENSES";
      rs = sqlStatement.executeQuery (query);
      SQLResultSetRowData rowData = new SQLResultSetRowData (rs);
      HTMLTable[] generatedHtml = converter.convertToTables(rowData);
      grid.addElement(generatedHtml[0]);
    }
    sqlStatement.close();
    // Примечание: Оператор close должен быть выполнен только после получения таблицы результатов.

    grid.addElement(new SubmitFormInput("returningToMain", "В начало"));

    mainForm.addElement(grid);
    page.append(mainForm.toString());
  }
```

```
  catch (Exception e) { e.printStackTrace (); }
  page.append("</body></html>");
  return page.toString();
}


private String showHtmlForListingAllReported ()
{
  String title = "Все автомобили, о которых есть сведения";
  StringBuffer page = new StringBuffer();
  page.append (showHeader (title));

  try
  {
    // Создание объекта формы HTML.
    HTMLForm form = new HTMLForm("LightsOn");

    // Создание одной панели, на которой будут расположены
    // созданные элементы HTML.
    GridLayoutFormPanel grid = new GridLayoutFormPanel();

    // Выбор расположения элементов созданной таблицы.
    HTMLTable table = new HTMLTable();
    table.setAlignment(HTMLConstants.LEFT);
    table.setBorderWidth(3);

    // Создание и добавление названия и заголовка таблицы.
    HTMLTableCaption caption = new HTMLTableCaption();
    caption.setAlignment(HTMLConstants.TOP);
    caption.setElement(title);
    table.setCaption(caption);
    table.setHeader(new String[] { "Знак", "Цвет", "Класс", "Дата", "Время" } );

    // Создание программы преобразования, формирующей таблицу HTML из таблицы
    // результатов, полученной в результате обработки запроса к базе данных.
    HTMLTableConverter converter = new HTMLTableConverter();
    converter.setTable(table);

    getDatabaseConnection ();
    Statement sqlStatement = databaseConnection_.createStatement();

    // Проверка, не пуста ли база данных.
    String query = "SELECT COUNT(*) FROM REPORTS";
    ResultSet rs = sqlStatement.executeQuery (query);
    rs.next();  // поместить курсор в первую строку
    int rowCount = rs.getInt(1);

    if (rowCount == 0) {
      page.append ("<font size=4 color=red>No vehicles have been reported.</font>");
    }
    else {
      query = "SELECT LICENSE,COLOR,CATEGORY,DATE_ADDED,TIME_ADDED FROM REPORTS";
      rs = sqlStatement.executeQuery (query);
      SQLResultSetRowData rowData = new SQLResultSetRowData (rs);
      HTMLTable[] generatedHtml = converter.convertToTables(rowData);
      grid.addElement(generatedHtml[0]);
    }
    sqlStatement.close();
    // Примечание: Оператор close должен быть выполнен только после получения таблицы результатов.

    grid.addElement(new SubmitFormInput("returningToMain", "В начало"));
    form.addElement(grid);
    page.append(form.toString());
  }
  catch (Exception e) { e.printStackTrace (); }
  page.append("</body></html>");
```

```
    return page.toString();
  }

}
```

Примеры простых программ

В данном разделе приведены простые примеры, иллюстрирующие некоторые способы создания программ на Java с помощью классов IBM Toolbox for Java. Эти примеры снабжены подробными пояснениями, что позволяет их использовать даже при отсутствии опыта работы с классами IBM Toolbox for Java.

If you want some help getting started, see Writing your first IBM Toolbox for Java program.

Ссылки на многие другие примеры IBM Toolbox for Java приведены в разделе Примеры кода.

Примеры простых программ разбиты по следующим категориям:

- Вызов команд
- Работа с очередями сообщений
- Работа с файлами на уровне записей
- Создание и заполнение таблиц с помощью классов JDBC
- Просмотр списка заданий сервера в окне графического интерфейса

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

All programs contained herein are provided to you ″AS IS″ without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Создание первой программы с помощью IBM Toolbox for Java

Для выполнения этого простого упражнения нужно установить Java на рабочей станции.

При выборе версии Java ознакомьтесь с требованиями из раздела Условия выполнения программ на Java.

После установки Java на клиенте нужно выполнить следующие действия:

1. Скопируйте файл jt400.jar на рабочую станцию.
2. Добавьте полный путь к файлу jt400.jar в переменную CLASSPATH. Например, если файл jt400.jar находится в каталоге c:\lib рабочей станции с операционной системой Windows, добавьте следующую строку в переменную CLASSPATH:

   ```
   ;c:\lib\jt400.jar
   ```
3. Запустите текстовый редактор и введите первый пример простой программы

   Примечание: Текст рекомендуется ввести со всеми примечаниями (такими как Примечание 1, Примечание 2 и т.д.). Сохраните файл с именем CmdCall.java.
4. Запустите сеанс командной строки на рабочей станции и введите следующую команду для компиляции примера программы:

   ```
   javac CmdCall.java
   ```
5. В сеансе командной строки введите следующую команду для выполнения примера:

```
java CmdCall
```

[Примеры простых программ]

Пример: применение объекта CommandCall

This example uses the IBM Toolbox for Java access class CommandCall.

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// This source is an example of IBM Toolbox for Java "Job List".
//
////////////////////////////////////////////////////////////////////////////
//
// The access classes of IBM Toolbox for Java are in the
// com.ibm.as400.access.package. Для работы с классами IBM Toolbox for
// Java этот пакет необходимо импортировать.
//
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;

public class CmdCall
{
   public static void main(String[] args)
   {
      // Как и другие классы Java, классы IBM Toolbox for Java
      // выбрасывают исключения при ошибках и сбоях. Их необходимо
      // отлавливать с помощью программ, использующих IBM Toolbox for Java.
      try Примечание 1
      {
         AS400 system = new AS400();

         CommandCall cc = new CommandCall(system); Примечание 2

         cc.run("CRTLIB MYLIB"); Примечание 3

         AS400Message[] ml = cc.getMessageList(); Примечание 4

         for (int i=0; i<ml.length; i++)
         {
            System.out.println(ml[i].getText()); Примечание 5
         }
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }

      System.exit(0);
   }
}
```

1. Для идентификации целевого сервера IBM Toolbox for Java применяет объект AS400. Если объект AS400 создан без указания параметров, IBM Toolbox for Java предложит вам указать имя системы, а также ИД пользователя и пароль. В класс AS400 входит конструктор, использующий имя системы, ИД пользователя и пароль.
2. Для передачи команд на сервер применяется объект CommandCall IBM Toolbox for Java. Объект CommandCall передается объекту AS400. Таким образом становится известно, на каком сервере должна выполняться команда.

3. Для выполнения команды нужно вызвать метод run().
4. Результатом выполнения является список сообщений i5/OS. IBM Toolbox for Java представляет эти сообщения как объекты AS400Message. Эти объекты хранятся в объекте CommandCall.
5. Печать текста сообщения. Помимо текста, доступны идентификаторы сообщений, сведения об их серьезности и прочая информация. Данная программа печатает только текст сообщений.

Пример: работа с сообщениями (часть 1 из 3)

This source is an example of IBM Toolbox for Java message queue.

[Следующая часть]

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с очередью сообщений с помощью IBM Toolbox for Java
//
// Пример применения класса MessageQueue IBM Toolbox for Java.
//
////////////////////////////////////////////////////////////////////////////

примеры пакетов;  Примечание 1


import java.io.*;
import java.util.*;

import com.ibm.as400.access.*;  Примечание 2

public class displayMessages extends Object
{

   public static void main(String[] parameters)  Примечание 3
   {
      displayMessages me = new displayMessages();
      me.Main(parameters);  Примечание 4

      System.exit(0);  Примечание 5
   }


   void displayMessage()
   {
   }


   void Main(String[] parms)
   {
      try  Примечание 6
      {

               // Код IBM Toolbox for Java
      }
      catch (Exception e)
      {
         e.printStackTrace();  Примечание 7
      }
   }
}
```

1. Данный класс входит в пакет 'examples'. Пакеты применяются в Java во избежание конфликтов между именами файлов классов Java.

2. В этой строке программе предоставляется доступ ко всем классам IBM Toolbox for Java из пакета access. Имена всех классов пакета access начинаются с префикса **com.ibm.as400**. После импорта пакета в программе не обязательно указывать префиксы перед именами классов. Например, класс AS400 можно вызывать как AS400 вместо com.ibm.as400.AS400.
3. У данного класса есть метод **main**, поэтому его можно запускать как приложение. Для запуска нужно выполнить команду **java examples.displayMessages**. Учтите, что при вызове программы учитывается регистр букв. Поскольку в данной программе применяется класс IBM Toolbox for Java, в переменной CLASSPATH должен быть указан путь к файлу jt400.zip.
4. Метод main (примечание 3) является статическим. Одно из ограничений в применении статических методов заключается в том, что они могут вызывать только другие статические методы из своего класса. Для того чтобы устранить это ограничение, многие программы на языке Java создают объект, а затем выполняют инициализацию с помощью метода **Main**. Метод Main() может вызывать любые методы объекта displayMessages.
5. Для выполнения операций IBM Toolbox for Java в IBM Toolbox for Java создаются нити. Поэтому для нормального завершения работы программа должна вызывать функцию **System.exit(0)**. Рассмотрим пример, когда программа была вызвана из командной строки DOS в операционной системе Windows 95. Если указанная функция не будет вызвана, то после завершения программы не появится приглашение командной строки, и пользователю придется нажать Ctrl-C, чтобы оно появилось.
6. В классах IBM Toolbox for Java применяются исключительные ситуации, которые должны обрабатываться в пользовательской программе.
7. Данная программа показывает текст исключительной ситуации во время обработки ошибок. Описания исключительных ситуаций IBM Toolbox for Java выводятся на том языке, который установлен на рабочей станции.

[Следующая часть]

Пример: работа с сообщениями (часть 2 из 3)

This source is an example of IBM Toolbox for Java message queue.

[Предыдущая часть | Следующая часть]

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с очередью сообщений с помощью IBM Toolbox for Java
//
// Пример применения класса MessageQueue IBM Toolbox for 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();  Примечание 1


                       if (parms.length > 0)
                          system.setSystemName(parms[0]);  Примечание 2

       }
       catch (Exception e)
       {
          e.printStackTrace();
       }
    }
}
```

1. Программа выбирает сервер, с которым нужно установить соединение, с помощью объекта **AS400**. За единственным исключением, все программы, работающие с ресурсами сервера, должны создавать этот объект. Исключение составляет JDBC. Если в вашей программе применяется JDBC, то драйвер JDBC IBM Toolbox for Java автоматически создаст для нее объект AS400.
2. Данная программа рассматривает первый параметр как имя сервера. Если вы вызовете программу с параметром, то имя системы AS/400 будет задано с помощью метода **setSystemName** объекта AS400. Кроме того, в объекте AS400 должна быть задана идентификационная информация для подключения к серверу:
   - Если программа будет запущена на рабочей станции, IBM Toolbox for Java предложит пользователю ввести ИД и пароль. **Примечание:** если программа будет вызвана без имени системы AS/400, то объект AS400 запросит его у пользователя.
   - If the program is running on the i5/OS JVM, then the user ID and password of the user running the Java program is used. Кроме того, пользователю не нужно будет указывать имя системы, поскольку программа выполняется в конкретной системе AS/400.

[Предыдущая часть | Следующая часть]

Пример: работа с сообщениями (часть 3 из 3)

This source is an example of IBM Toolbox for Java message queue.

[Предыдущая часть]

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с очередью сообщений с помощью IBM Toolbox for Java
//
// Пример применения класса MessageQueue IBM Toolbox for 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);  Примечание 1


                  Enumeration e = queue.getMessages();  Примечание 2

                  while (e.hasMoreElements())
                  {

                      QueuedMessage message = (QueuedMessage) e.nextElement();  Примечание 3
                      System.out.println(message.getText());  Примечание 4
                   }
               }
               catch (Exception e)
               {
                  e.printStackTrace();
               }
         }
}
```

1. Данная программа предназначена для просмотра сообщений, хранящихся в очередях сообщений сервера. The **MessageQueue** object of the IBM Toolbox for Java is used for this task. При создании объекта очереди сообщений в качестве параметров используются объект AS400 и имя очереди сообщений. Объект AS400 указывает, в каком сервере находится ресурс, а имя очереди сообщений задает конкретную очередь. В данной программе применяется константа, указывающая на очередь пользователя, работающего в системе.
2. Объект очереди сообщений получает список сообщений с сервера. Фактически соединение с сервером устанавливается только в этот момент.
3. Сообщение удаляется из списка. Теперь оно находится в объекте QueuedMessage IBM Toolbox for Java.
4. Печать текста сообщения.

[Предыдущая часть]

Пример: работа с файлами на уровне записей (часть 1 из 2)

This program will prompt the userfor the name of the server and the file to display. The file must exist and contain records. Each record in the file will be displayed to System.out.

[Следующая часть]

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с файлом на уровне записей. Эта программа предложит пользователю
// указать имя сервера и просматриваемый файл. Этот файл должен существовать и
// содержать записи. Каждая запись файла будет выведена в
// System.out.
//
// Формат вызова: java RLSequentialAccessExample
//
// Пример применения класса RecordLevelAccess IBM Toolbox for 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("Имя системы: ");
        systemName = inputStream.readLine();

        System.out.print("Библиотека, в которой расположен файл: ");
        library = inputStream.readLine();

        System.out.print("Имя файла: ");
        file = inputStream.readLine();

        System.out.print("Имя элемента (для чтения первого элемента нажмите Enter): ");
        member = inputStream.readLine();
        if (member.equals(""))
        {
          member = "*FIRST";
        }

        System.out.println ();


             }
```

```
catch (Exception e)
{
  System.out.println("Ошибка ввода пользователя.");
  e.printStackTrace();
  System.exit(0);
}


AS400 system = new AS400(systemName);  Примечание 1
try
{
  system.connectService(AS400.RECORDACCESS);
}
catch(Exception e)
{
  System.out.println("Не удалось создать соединение с доступом на уровне записей.");
  System.out.println("Специальные указания по созданию соединений на уровне записей
    приведены в файле установки руководства по программированию");
  e.printStackTrace();
  System.exit(0);
}


QSYSObjectPathName filePathName = new QSYSObjectPathName(library, file, member, "MBR");
                                                          Примечание 2


SequentialFile theFile = new SequentialFile(system, filePathName.getPath());  Примечание 3


AS400FileRecordDescription recordDescription =
     new AS400FileRecordDescription(system, filePathName.getPath());
try
{
  RecordFormat[] format = recordDescription.retrieveRecordFormat();  Примечание 4


  theFile.setRecordFormat(format[0]);  Примечание 5


  theFile.open(AS400File.READ_ONLY, 100, AS400File.COMMIT_LOCK_LEVEL_NONE);  Примечание 6

  System.out.println("Просмотр файла " + library.toUpperCase() + "/" +
    file.toUpperCase() + "(" + theFile.getMemberName().trim() + "):");

  Record record = theFile.readNext();  Примечание 7
  while (record != null)
  {
    System.out.println(record);
    record = theFile.readNext();
  }
  System.out.println ();




  theFile.close();  Примечание 8


  system.disconnectService(AS400.RECORDACCESS);  Примечание 9
}
catch (Exception e)
{
  System.out.println("При попытке просмотра файла произошла ошибка.");
  e.printStackTrace();

  try
```

```
        {
          // Закрыть файл
          theFile.close();
        }
        catch(Exception x)
        {
        }

        system.disconnectService(AS400.RECORDACCESS);
        System.exit(0);
      }

      // Убедитесь, что приложение завершило работу (см. файл readme)
      System.exit(0);
    }
  }
```

1. Эта команда создает объект AS400 и устанавливает соединение для доступа на уровне записей.
2. Эта команда создает объект QSYSObjectPathName, в котором будет храниться путь к объекту в IFS.
3. Этот оператор создает объект, соответствующий существующему файлу сервера с последовательным доступом. Содержимое данного файла будет показано на экране.
4. Эти команды позволяют получить формат записей файла.
5. Эта команда задает формат записи файла.
6. Эта команда открывает выбранный файл для чтения. За один проход будет считываться по 100 записей (если это возможно).
7. Эта команда считывает очередную запись.
8. Эта команда закрывает файл.
9. Эта команда закрывает соединение с доступом на уровне записей.

[Следующая часть]

Пример: работа с файлами на уровне записей (часть 2 из 2)

This example demonstrates the use of record level access.

[Предыдущая часть]

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример работы с файлом на уровне записей.
//
// Формат вызова: 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";  Примечание 1

    try
    {
      SequentialFile theFile = new SequentialFile(system, filePathName);

      // Начало второго примечания
      CharacterFieldDescription lastNameField =
         new CharacterFieldDescription(new AS400Text(20), "LNAME");
      CharacterFieldDescription firstNameField =
         new CharacterFieldDescription(new AS400Text(20), "FNAME");
      BinaryFieldDescription yearsOld =
         new BinaryFieldDescription(new AS400Bin4(), "AGE");

      RecordFormat fileFormat = new RecordFormat("RF");
      fileFormat.addFieldDescription(lastNameField);
      fileFormat.addFieldDescription(firstNameField);
      fileFormat.addFieldDescription(yearsOld);

      theFile.create(fileFormat, "Файл с именами и значениями возраста");  Примечание 2
      // Конец второго примечания


      theFile.open(AS400File.READ_WRITE, 1, AS400File.COMMIT_LOCK_LEVEL_NONE);

      // Начало третьего примечания
      Record newData = fileFormat.getNewRecord();
      newData.setField("LNAME", "Доу");
      newData.setField("FNAME", "Джон");
      newData.setField("AGE", new Integer(63));

      theFile.write(newData);  Примечание 3
      // Конец третьего примечания

      theFile.close();
    }
    catch(Exception e)
    {
      System.out.println("Произошла ошибка: ");
      e.printStackTrace();
    }

    system.disconnectService(AS400.RECORDACCESS);

    System.exit(0);
  }
}
```

1. (args[0]) в предыдущей строке и MYFILE.FILE необходимы для выполнения остальной части примера. В данной программе предполагается, что на сервере есть библиотека MYLIB, и у вас есть права доступа к ней.
2. Команды, расположенные между строками комментария Java Начало второго комментария и Конец второго комментария, иллюстрируют создание собственного формата записи (как альтернативу существующему формату записи, получаемому из файла). Последняя строка этого кода создает файл на сервере.
3. Команды между строками комментария Java Начало третьего комментария и Конец третьего комментария иллюстрируют процесс создания записи с последующим сохранением в файле.

[Предыдущая часть]

Пример: создание и заполнение таблицы с помощью классов JDBC (часть 1 из 2)

This program uses the IBM Toolbox for Java JDBC driver to create and populate a table.

[Следующая часть]

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример применения JDBCPopulate.  This program uses the IBM Toolbox for Java JDBC driver
// с помощью драйвера JDBC.
//
// Формат вызова:
//    JDBCPopulate система имя-набора имя-таблицы
//
// Пример:
//    JDBCPopulate MySystem MyLibrary MyTable
//
// Эта программа - пример работы с драйвером JDBC IBM Toolbox for Java.
//
////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCPopulate
{

    private static final String words[]
        = { "One",      "Two",      "Three",    "Four",     "Five",
            "Six",      "Seven",    "Eight",    "Nine",     "Ten",
            "Eleven",   "Twelve",   "Thirteen", "Fourteen", "Fifteen",
            "Sixteen",  "Seventeen","Eighteen", "Nineteen", "Twenty" };

    public static void main (String[] parameters)
    {

        if (parameters.length != 3) {
            System.out.println("");
            System.out.println("Формат:");
            System.out.println("");
            System.out.println("   JDBCPopulate system collectionName tableName");
            System.out.println("");
            System.out.println("");
            System.out.println("Например:");
            System.out.println("");
            System.out.println("");
            System.out.println("   JDBCPopulate MySystem MyLibrary MyTable");
            System.out.println("");
            return;
        }

        String system           = parameters[0];
        String collectionName   = parameters[1];
        String tableName        = parameters[2];

        Connection connection   = null;

        try {


            DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());  Примечание 1
```

```
            connection = DriverManager.getConnection ("jdbc:as400://"
                + system + "/" + collectionName);  Примечание 2

            try {
                Statement dropTable = connection.createStatement ();
                dropTable.executeUpdate ("DROP TABLE " + tableName);  Примечание 3
            }
            catch (SQLException e)
        {
            }


            Statement createTable = connection.createStatement ();
            createTable.executeUpdate ("Создать таблицу " + tableName
                + " (I INTEGER, WORD VARCHAR(20), SQUARE INTEGER, "
                + " SQUAREROOT DOUBLE)");  Примечание 4


            PreparedStatement insert = connection.prepareStatement ("Добавить в "
                + tableName + " (I, WORD, SQUARE, SQUAREROOT) "
                + " VALUES (?, ?, ?, ?)");  Примечание 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 ();  Примечание 6
            }

            System.out.println ("В таблицу " + collectionName + "." + tableName
                + " внесены данные.");
        }

        catch (Exception e) {
            System.out.println ();
            System.out.println ("Ошибка: " + e.getMessage());
        }

        finally
        {

            try {
                if (connection != null)
                    connection.close ();  Примечание 7
            }
            catch (SQLException e)
        {
                // Игнорировать.
            }
        }

        System.exit (0);
    }

}
```

1. В этой строке выполняется загрузка драйвера JDBC IBM Toolbox for Java. Драйвер JDBC обеспечивает взаимодействие JDBC и базы данных, с которой вы работаете.
2. Этот оператор устанавливает соединение с базой данных. Пользователю предлагается ввести свой ИД и пароль. Предоставляется стандартная схема, поэтому вам не нужно указывать имя таблицы в операторах SQL.
3. Эти команды удаляют таблицу в случае, если она уже существует.
4. С помощью этих команд создается таблица.

5. С помощью этой команды подготавливается оператор, который будет добавлять ряды данных в таблицу. Поскольку данный оператор будет выполняться несколько раз, целесообразно подготовить его и в дальнейшем вызывать с помощью PreparedStatement и маркеров параметров.
6. С помощью этого раздела кода данные вносятся в таблицу; при каждом проходе этого цикла в таблицу добавляется новая строка.
7. Теперь, когда таблица создана и заполнена, данный оператор завершает соединение с базой данных.

[Следующая часть]

Пример: создание и заполнение таблицы с помощью классов JDBC (часть 2 из 2)

This program uses the IBM Toolbox for Java JDBC driver to query a table and output its contents.

[Предыдущая часть]

Этим примером можно пользоваться в качестве шаблона при разработке реальной программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример применения JDBCQuery.  Эта программа с помощью драйвера JDBC IBM Toolbox for Java
// отправляет запрос в таблицу и выводит ее содержимое.
//
// Формат вызова:
//    JDBCQuery система имя-набора имя-таблицы
//
// Пример:
//    JDBCQuery MySystem qiws qcustcdt
//
// Эта программа - пример работы с драйвером JDBC IBM Toolbox for Java.
//
////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class JDBCQuery
{

    // Форматирование строки по заданной ширине.
    private static String format (String s, int width)
    {
        String formattedString;

        // Если строка короче, чем требуется,
        // ее нужно дополнить пробелами
        if (s.length() < width) {
            StringBuffer buffer = new StringBuffer (s);
            for (int i = s.length(); i < width; ++i)
                buffer.append (" ");
            formattedString = buffer.toString();
        }

        // В противном случае строку нужно усечь.
        else
            formattedString = s.substring (0, width);

        return formattedString;
    }

    public static void main (String[] parameters)
    {
        // Проверка входных параметров.
```

```
if (parameters.length != 3) {
    System.out.println("");
    System.out.println("Формат:");
    System.out.println("");
    System.out.println("   JDBCQuery system collectionName tableName");
    System.out.println("");
    System.out.println("");
    System.out.println("Например:");
    System.out.println("");
    System.out.println("");
    System.out.println("   JDBCQuery mySystem qiws qcustcdt");
    System.out.println("");
    return;
}

String system           = parameters[0];
String collectionName   = parameters[1];
String tableName        = parameters[2];

Connection connection   = null;

try {

    DriverManager.registerDriver(new com.ibm.as400.access.AS400JDBCDriver());  Примечание 1

    // Создание соединения с базой данных.  Поскольку ИД пользователя
    // и пароль заранее не известны, на экране появится приглашение.
    connection = DriverManager.getConnection ("jdbc:as400://" + system);
    DatabaseMetaData dmd = connection.getMetaData ();  Примечание 2

    // Выполнение запроса.
    Statement select = connection.createStatement ();
    ResultSet rs = select.executeQuery ("SELECT * FROM "
        + collectionName + dmd.getCatalogSeparator() + tableName);  Примечание 3

    // Получение информации о результатах.  Задание
    // ширины колонки равной максимальному из двух значений:
    // длины метки и длины данных.
    ResultSetMetaData rsmd = rs.getMetaData ();
    int columnCount = rsmd.getColumnCount (); Примечание 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));  Примечание 5
    }

    // Вывод заголовков столбцов.
    for (int i = 1; i <= columnCount; ++i) {
        System.out.print (format (rsmd.getColumnLabel(i), columnWidths[i-1]));
        System.out.print("   ");
    }
    System.out.println ();

    // Вывод пунктирной линии.
    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 ();

    // Итерационный блок вывода колонок с результатами
    // для каждого ряда данных.
    while (rs.next ()) {
```

```
                for (int i = 1; i <= columnCount; ++i) {
                    String value = rs.getString (i);
                    if (rs.wasNull ())
                        value = "<null>";  Note 6
                    System.out.print (format (value, columnWidths[i-1]));
                    System.out.print("   ");
                }
                System.out.println ();
            }

        }

        catch (Exception e) {
            System.out.println ();
            System.out.println ("Ошибка: " + e.getMessage());
        }

        finally
        {

            // Очистка.
            try {
                if (connection != null)
                    connection.close ();
            }
            catch (SQLException e)
        {
                // Игнорировать.
            }
        }

        System.exit (0);
    }

}
```

1. В этой строке выполняется загрузка драйвера JDBC IBM Toolbox for Java. Драйвер JDBC обеспечивает взаимодействие JDBC и базы данных, с которой вы работаете.
2. Эта команда получает метаданные для соединения - объект, задающий большинство характеристик базы данных.
3. Этот оператор выполняет запрос для указанной таблицы.
4. Эти команды служат для получения сведений о таблице.
5. Эти операторы задают ширину столбца равным максимальному из двух значений: длины метки и длины данных.
6. В этом цикле на экран выводится содержимое всех строк и столбцов таблицы.

[Предыдущая часть]

Пример: просмотр списка заданий сервера в GUI

This example uses the IBM Toolbox for Java VJobList class.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Example using the IBM Toolbox for Java vaccess
//
//
// This source is an example of IBM Toolbox for Java "Job List".
//
////////////////////////////////////////////////////////////////////////////

примеры пакетов;  Примечание 1
```

```
import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;  Примечание 2

import javax.swing.*;  Примечание 3
import java.awt.*;
import java.awt.event.*;

public class GUIExample
{

  public static void main(String[] parameters)  Примечание 4
  {
     GUIExample example = new GUIExample(parameters);

  }

  public GUIExample(String[] parameters)
  {
     try  Примечание 5
     {
          // Создание объекта AS400.
     // Первый аргумент - имя системы.
          AS400 system = new AS400 (parameters[0]);  Примечание 6

          VJobList jobList = new VJobList (system);  Примечание 7

     // Создание фрейма.
     JFrame frame = new JFrame ("Пример списка заданий");  Примечание 8

     // Создание адаптера окна ошибок. В этом окне будут показаны сведения об ошибках.
ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (frame);  Примечание 9

     // Создание панели проводника для просмотра списка заданий.
     AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList);  Примечание 10

     explorerPane.addErrorListener (errorHandler);  Примечание 11

     // Загрузка информации из системы методом load.
     explorerPane.load();  Примечание 12

     // При закрытии окна работа программы завершается.
     frame.addWindowListener (new WindowAdapter ()  Примечание 13
     {
           public void windowClosing (WindowEvent event)
           {
              System.exit(0);
           }
     } );

     // Размещение фрейма с панелью проводника.
     frame.getContentPane().setLayout(new BorderLayout() );
     frame.getContentPane().add("Center", explorerPane);  Примечание 14

     frame.pack();
     frame.show();  Примечание 15
     }

     catch (Exception e)
     {
          e.printStackTrace();  Примечание 16
     }
     System.exit(0);  Примечание 17

     }
}
```

1. Данный класс входит в пакет 'examples'. Пакеты применяются в Java во избежание конфликтов между именами файлов классов Java.
2. Эта строка делает доступными все классы пакета vaccess IBM Toolbox for Java. Имена всех классов пакета vaccess начинаются с префикса com.ibm.as400.vaccess. После импорта пакета для вызова входящих в него классов можно указывать только их имена (без имени пакета). Например, класс AS400ExplorerPane можно вызвать по имени AS400ExplorerPane вместо com.ibm.as400.AS400ExplorerPane.
3. Эта строка делает доступными все классы Java Foundation Classes в пакете Swing. Для применения классов GUI Vaccess IBM Toolbox for Java программам на Java требуются JDK 1.1.2 и Java Swing 1.0.3 фирмы Sun Microsystems, Inc. Пакет Swing входит в состав продукта JFC 1.1 фирмы Sun.
4. У данного класса есть метод main; поэтому его можно запускать как приложение. Для запуска нужно выполнить команду ″java examples.GUIExample имя_сервера″, где имя_сервера - это имя сервера. Для работы программы необходимо, чтобы в переменной CLASSPATH был указан путь к файлу jt400.zip или jt400.jar.
5. В классах IBM Toolbox for Java применяются исключительные ситуации, которые должны обрабатываться в пользовательской программе.
6. Класс AS400 применяется в IBM Toolbox for Java. Он управляет информацией входа в систему, устанавливает и поддерживает соединения через API сокетов и отвечает за обмен данными. В данном примере в объект AS/400 передается имя сервера.
7. Класс VJobList применяется в IBM Toolbox for Java для представления списка заданий сервера, который может быть показан с помощью компонента графического интерфейса (GUI). Учтите, что сервер, к которому относится список, задается с помощью объекта AS400.
8. В этой строке создается фрейм (окно верхнего уровня), в котором будет показан список заданий.
9. ErrorDialogAdapter - это компонент GUI IBM Toolbox for Java, предназначенный для автоматического вывода окна в случае возникновения ошибки.
10. В этой строке создается AS400ExplorerPane - компонент GUI, представляющий иерархическую структуру объектов в ресурсе сервера. В левой части панели AS400ExplorerPane показано дерево объектов, причем корень дерева соответствует объекту VJobList, а в правой части - сведения о ресурсе. Данная строка только инициализирует панель; она не загружает в нее содержимое VJobList.
11. Эта строка активизирует обработчик ошибок, созданный на строке 9 и предназначенный для контроля компонента GUI VJobList.
12. Эта строка загружает содержимое JobList в ExplorerPane. Для обмена данными с сервером и загрузки информации из него необходимо явно вызвать этот метод. В этом случае программа может управлять обменом данными с сервером, что предоставляет следующие возможности:
    - Загрузка содержимого до добавления панели во фрейм. Фрейм будет показан только после того, как в него будет загружена вся информация (как в данном примере).
    - Загрузка содержимого после добавления панели во фрейм и вывода фрейма. Курсор во фрейме примет форму, означающую, что система занята, а информация будет выводиться по мере загрузки.
13. Эта строка добавляет объект контроля за окном, завершающий работу приложения, когда пользователь закрывает фрейм.
14. Эта строка добавляет компонент GUI со списком заданий в центр управляющего фрейма.
15. Эта строка вызывает метод, который делает окно видимым для пользователя.
16. Информация об исключительных ситуациях IBM Toolbox for Java выводится на том языке, который применяется на рабочей станции. Например, данная программа показывает текст исключительной ситуации во время обработки ошибки.
17. Для выполнения операций IBM Toolbox for Java в IBM Toolbox for Java создаются нити. Поэтому для нормального завершения работы программа должна вызывать функцию System.exit(0). Если программа не вызывает эту функцию, то в случае ее запуска из командной строки DOS в Windows 95 после завершения работы программы не появится приглашение командной строки.

Примеры: Советы программисту

This topic lists the code examples that are provided throughout the documentation of IBM Toolbox for Java tips for programming.

Управление соединениями

- Example: Making a connection to the system with a CommandCall object
- Example: Making two connections to the system with a CommandCall object
- Пример: Создание объектов CommandCall и IFSFileInputStream objects с помощью объекта AS400
- Example: Using AS400ConnectionPool to preconnect to the system
- Example: Using AS400ConnectionPool to preconnect to a specific service on the system, then reuse the connection

Подключение и отключение

- Example: How a Java program preconnects to the system
- Example: How a Java program disconnects from the system
- Example: How a Java program disconnects and reconnects to the system with disconnectService() and run()
- Example: How a Java program disconnects from the system and fails to reconnect

Исключительные ситуации

- Пример: Использование исключительных ситуаций

События при ошибках

- Пример: Обработка событий, связанных с ошибками
- Пример: Определение обработчика ошибок
- Пример: Использование собственного обработчика ошибок

Класс Trace

- Пример: Использование трассировки
- Пример: Использование setTraceOn()
- Пример: Применение трассировки отдельных компонентов

Оптимизация

- Пример: Создание двух объектов AS400
- Пример: Представление второго сервера с помощью объекта AS400

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Examples: ToolboxME

This section lists the code examples that are provided throughout the IBM Toolbox for Java 2 Micro Edition documentation.

- “ToolboxME example: JdbcDemo.java” на стр. 367
- “Example: Using ToolboxME, MIDP, and JDBC”
- “Example: Using ToolboxME, MIDP, and IBM Toolbox for Java” на стр. 711

The following disclaimer applies to all of the IBM Toolbox for Java examples:

Example: Using ToolboxME, MIDP, and JDBC

The following source illustrates one way that your IBM Toolbox for Java 2 Micro Edition application can use the Mobile Information Device Profile (MIDP) and JDBC to access a database and store information offline.

Программа, рассмотренная в данном примере, предназначена для агента по продаже недвижимости, которому необходимо просматривать и делать заявки на объекты, выставленные на продажу. The agent uses a Tier0 device to access information for the properties, which is stored in the server database.

Рабочая программа, получаемая в результате преобразования исходного кода, подключается к базе данных, созданной специально для этой цели.

Для преобразования исходного кода в рабочую версию и получения исходного кода, на основе которого вы создадите и заполните базу данных, вы должны загрузить пример. Кроме того, вам рекомендуется ознакомиться с инструкциями по созданию и запуску примера программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// ToolboxME example. This program is an example MIDlet that shows how
// каким образом можно написать приложение JdbcMe для профайла MIDP.
// Информация об обработке каждого запрошенного перехода приведена в описании методов
// startApp, pauseApp, destroyApp и commandAction.
//
////////////////////////////////////////////////////////////////////////////

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 = "Нет заявок, выберите данный пункт для загрузки заявок";
private List      main = new List("Демонстрационная версия заявок 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("текст_заявок", "", 10, TextField.NUMERIC);
private Form      errorForm = null;

private Command exitCommand   = new Command("Выход", Command.SCREEN, 0);
private Command backCommand   = new Command("Назад", Command.SCREEN, 0);
private Command cancelCommand = new Command("Отмена", Command.SCREEN, 0);
private Command goCommand     = new Command("Далее", Command.SCREEN, 1);
private Displayable   onErrorGoBackTo = null;

/*
 * Создание нового объекта JdbcMidpBid.
 */
public JdbcMidpBid()
{
    display = Display.getDisplay(this);
}

/**
 * Вывод главного меню
 */
public void startApp()
{
    main.append("Показать заявки", null);
    main.append("Получить новые заявки", null);
    main.append("Параметры", null);
    main.append("О программе", null);
    main.addCommand(exitCommand);
    main.setCommandListener(this);

    display.setCurrent(main);
}

public void commandAction(Command c, Displayable s)
{
    // Обработка всех команд exitCommand одинакова.
    if (c == exitCommand)
    {
        destroyApp(false);
        notifyDestroyed();
        return;
    }
    if (s instanceof List)
    {
        List     current = (List)s;

        // Действие произошло на главной странице
        if (current == main)
        {
            int   idx = current.getSelectedIndex();
            switch (idx)
```

```
                {
                case 0:     // Показать текущие заявки
                    showBids();
                    break;
                case 1:     // Получить новые заявки
                    getNewBids();
                    break;
                case 2:     // Параметры
                    doSettings();
                    break;
                case 3:     // О программе
                    aboutBox();
                    break;
                default :
                    break;
                }
                return;
            } // current == main

            // Действие произошло на странице распечаток
            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";
                    // Также отслеживать текущую строку автономного
                    // набора результатов. Она оказывается совпадающей
// с индексом в списке.
                    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)
                {
                    // Обработка окна Параметры закончена.
                    display.setCurrent(main);
                    settingsForm = null;
                    return;
```

```
            }
        } // settingsForm
        if (current == aboutBox)
        {
            if (c == backCommand)
            {
                // Обработка окна О программе закончена.
                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)
                {
                    // Если текущая позиция уже не на
                    // bidForm, то аннулировать bidForm.
                    bidForm = null;
                }
                return;
            }
            return;
        } // current == bidForm
    } // instanceof Form
}

public void aboutBox()
{
    aboutBox = new Form("Окно О программе");
    aboutBox.setTitle("О программе");
    aboutBox.append(new StringItem("", "пример Midp RealEstate для JdbcMe "));
    aboutBox.addCommand(backCommand);
    aboutBox.setCommandListener(this);
    display.setCurrent(aboutBox);
}

/**
 *  Форма параметров.
 */
public void doSettings()
{
    settingsForm = new Form("Форма_параметров");
    settingsForm.setTitle("Параметры");
    settingsForm.append(new StringItem("", "URL базы данных"));
    settingsForm.append(urlText);
    settingsForm.append(new StringItem("", "Сервер JdbcMe"));
    settingsForm.append(jdbcmeText);
    settingsForm.append(new StringItem("", "Трассировка"));

    settingsForm.addCommand(backCommand);
    settingsForm.setCommandListener(this);
    display.setCurrent(settingsForm);
}


/**
 * Показать меню заявок для выбранной целевой
```

```
 * заявки.
 */
public void bidOnProperty()
{
    StringItem item = new StringItem("", bidTarget);

    bidText = new TextField("текст_заявок", "", 10, TextField.NUMERIC);
    bidText.setString("");

    bidForm = new Form("форма_заявки");
    bidForm.setTitle("Отправить заявку для:");
    BID_PROPERTY = 0;
    bidForm.append(item);
    bidForm.append(new StringItem("", "Ваша заявка"));
    bidForm.append(bidText);
    bidForm.addCommand(cancelCommand);
    bidForm.addCommand(goCommand);
    bidForm.setCommandListener(this);
    display.setCurrent(bidForm);
}

/**
 * Вывести в меню распечаток текущий
 * список заявок, которые необходимо обработать.
 */
public void getNewBids()
{
    // Сброс старой распечатки
    listings = null;
    listings = new List("Заявки JdbcMe", Choice.IMPLICIT);
    java.sql.Connection     conn = null;
    Statement               stmt = null;
    try
    {
        conn = DriverManager.getConnection(urlText.getString() + ";" + jdbcmeText.getString());

        stmt = conn.createStatement();

        // Поскольку хранение подготовленных операторов нежелательно,
        // в этой среде лучше применить обычный оператор.
        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("Заявки не найдены", null);
            listings.addCommand(backCommand);
            listings.setCommandListener(this);
        }
    }
    catch (Exception e)
    {
        // В настоящий момент допустимых распечаток не получено,
        // поэтому выполняется сброс к пустому значению.
        listings = new List("Заявки JdbcMe", Choice.IMPLICIT);
        listings.append(GETBIDS, null);
        listings.addCommand(backCommand);
        listings.setCommandListener(this);

        // Возврат в главное меню после показа информации об ошибке.
```

```
            showError(main, e);
            return;
        }
        finally
        {
            if (conn != null)
            {
                try
                {
                    conn.close();
                }
                catch (Exception e)
                {
                }
            }
            conn = null;
            stmt = null;
        }
        showBids();
    }

    public void setupListingsFromOfflineData()
    {
        // Пропустить первые четыре строки в хранилище записей
        // (типы eyecatcher, version, num columns, sql column)
        //
        // Каждая последующая строка в хранилище записей
        // состоит из одного столбца. Запрос возвращает три
// столбца, которые будут переданы в виде одной объединенной строки.
        ResultSet         rs = null;
        listings.addCommand(backCommand);
        listings.setCommandListener(this);
        try
        {
            int                     i = 5;
            int                     max = 0;
            StringBuffer   buf  = new StringBuffer(20);

            // Creator и dbtype не используются в MIDP
            rs = new JdbcMeOfflineResultSet("JdbcMidpBidListings", 0, 0);
            if (rs == null)
            {
                // Новые распечатки...
                listings = new List("Заявки 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("Заявки не найдены", null);
            return;
        }
    }
    catch (Exception e)
    {
        // В настоящий момент допустимых распечаток не получено,
        // поэтому выполняется сброс к пустому значению.
        listings = new List("Заявки JdbcMe", Choice.IMPLICIT);
        listings.append(GETBIDS, null);
        listings.addCommand(backCommand);
        listings.setCommandListener(this);

        // Возврат в главное меню после показа информации об ошибке.
        showError(main, e);
        return;
    }
    finally
    {
        if (rs != null)
        {
            try
            {
                rs.close();
                            }
            catch (Exception e)
            {
            }
            rs = null;
        }
        System.gc();
    }
}

/**
 * Вывести в меню распечаток текущий
 * список заявок, которые необходимо обработать.
 */
public void submitBid()
{
    java.sql.Connection     conn = null;
    Statement               stmt = null;
    try
    {
        conn = DriverManager.getConnection(urlText.getString() + ";" + jdbcmeText.getString());

        stmt = conn.createStatement();

        // Поскольку хранение подготовленных операторов нежелательно,
        // в этой среде лучше применить обычный оператор.
        StringBuffer   buf = new StringBuffer(100);
        buf.append("Обновить QJdbcMe.RealEstate Установить текущую заявку = ");
        buf.append(bidText.getString());
        buf.append(" Где MLS = '");
        buf.append(bidTargetKey);
        buf.append("' и текущая заявка < ");
        buf.append(bidText.getString());
        String      sql = buf.toString();

        int   updated = stmt.executeUpdate(sql);
        if (updated == 1)
```

```
        {
            // Заявка принята.
            String oldS = listings.getString(bidRow);
            int   commaIdx = bidTarget.indexOf(',');
            String bidAddr = bidTarget.substring(0, commaIdx);

            String newS = bidTargetKey + "," + bidAddr + ", $" + bidText.getString();

            ResultSet         rs = null;
            try
            {
                // Creator и dbtype не используются в 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();
                            }

            // Также обновить текущий список для набора результатов.
            listings.set(bidRow, newS, null);
            display.setCurrent(listings);
            conn.commit();
        }
        else
        {
            conn.rollback();
            throw new SQLException("Не удалось отправить заявку, она была отправлена ранее");
        }
    }
    catch (SQLException e)
    {
        // Возврат в форму заявок после показа информации об ошибке.
        showError(bidForm, e);
        return;
    }
    finally
    {
        if (conn != null)
        {
            try
            {
                conn.close();
            }
            catch (Exception e)
            {
            }
        }
        conn = null;
        stmt = null;
    }

    // Выход без исключительной ситуации, затем показ текущих заявок
    showBids();
}

/**
 * Показать причину ошибки.
 */
public void showError(Displayable d, Exception e)
{
    String s = e.toString();
```

```
            onErrorGoBackTo = d;
            errorForm = new Form("Ошибка");
            errorForm.setTitle("Ошибка SQL");
            errorForm.append(new StringItem("", s));
            errorForm.addCommand(backCommand);
            errorForm.setCommandListener(this);
            display.setCurrent(errorForm);
        }

        /**
         * Показать текущие заявки.
         */
        public void showBids()
        {
            if (listings == null)
            {
                // Если текущих распечаток нет, настроить
                // их.
                listings = new List("Заявки JdbcMe", Choice.IMPLICIT);
                setupListingsFromOfflineData();
            }
            display.setCurrent(listings);
        }

        /**
         * Пауза в выполнении программы и освобождение неиспользуемой памяти.
         */
        public void pauseApp()
        {
            display.setCurrent(null);
        }

        /**
         * Общая очистка.
         */
        public void destroyApp(boolean unconditional)
        {
        }
    }
```

Информация, связанная с данной

“Профайл мобильных устройств (MIDP)” на стр. 354

Example: Using ToolboxME, MIDP, and IBM Toolbox for Java

The following source illustrates one way that your IBM Toolbox for Java 2 Micro Edition application can use the Mobile Information Device Profile (MIDP) and IBM Toolbox for Java to access System i data and services.

В этом примере демонстрируется работа всех функций, входящих в поддержку IBM Toolbox for Java 2 Micro Edition. На примере большого числа меню приложение показывает некоторые из множества различных способов применения этих функций устройством Tier0.

R When built as a working program, the example code below uses a Program Call Markup Language (PCML) file to run
R commands on the server.

Для преобразования исходного кода в рабочую версию и получения исходного кода PCML для запуска команд на сервере вы должны загрузить пример. Кроме того, вам рекомендуется ознакомиться с инструкциями по созданию и запуску примера программы.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
R ////////////////////////////////////////////////////////////////////////////
R //
R // ToolboxME example. This program is an example that shows how
```

```
R // ToolboxME can use PCML to System i access data and services.
R //
R // Для работы приложения необходимо, чтобы файл qsyrusri.pcml
R // был указан в разделе CLASSPATH сервера MEServer.
R //
R ////////////////////////////////////////////////////////////////////////////
R
R import java.io.*;
R import java.sql.*;
R import java.util.Hashtable;
R
R import javax.microedition.midlet.*;
R import javax.microedition.lcdui.*;
R import javax.microedition.rms.*;
R
R import com.ibm.as400.micro.*;
R
R
R public class ToolboxMidpDemo extends MIDlet implements CommandListener
R {
R     private Display   display_;
R
R     // Системный объект ToolboxME.
R     private AS400 system_;
R
R     private List      main_ = new List("Демонстрационная версия ToolboxME MIDP", Choice.IMPLICIT);
R
R     // Создание формы для каждого компонента.
R     private Form      signonForm_;
R     private Form      cmdcallForm_;
R     private Form      pgmcallForm_;
R     private Form      dataqueueForm_;
R     private Form      aboutForm_;
R
R     // Видимый текст для каждого компонента.
R     static final String SIGN_ON       = "Вход в систему";
R     static final String COMMAND_CALL  = "Вызов команды";
R     static final String PROGRAM_CALL  = "Вызов программы";
R     static final String DATA_QUEUE    = "Очередь данных";
R     static final String ABOUT         = "О программе";
R
R     static final String NOT_SIGNED_ON = "Не зарегистрирован в системе.";
R     static final String DQ_READ       = "Чтение";
R     static final String DQ_WRITE      = "Запись";
R
R     // Индикатор состояния входа в систему.
R     private Ticker    ticker_ = new Ticker(NOT_SIGNED_ON);
R
R     // Команды, которые можно выполнять.
R     private static final Command actionExit_   = new Command("Выход", Command.SCREEN, 0);
R     private static final Command actionBack_   = new Command("Назад", Command.SCREEN, 0);
R     private static final Command actionGo_     = new Command("Далее", Command.SCREEN, 1);
R     private static final Command actionClear_  = new Command("Очистить", Command.SCREEN, 1);
R     private static final Command actionRun_    = new Command("Выполнить", Command.SCREEN, 1);
R     private static final Command actionSignon_ = new Command(SIGN_ON, Command.SCREEN, 1);
R     private static final Command actionSignoff_= new Command("Выход из системы", Command.SCREEN, 1);
R
R     private Displayable   onErrorGoBackTo_;  // форма, возвращаемая по окончании
R                                              // просмотра формы ошибок
R
R     // Поля TextField для формы SignOn.
R     private TextField signonSystemText_ = new TextField("Система", "rchasdm3", 20,
R                                                       TextField.ANY);
R     private TextField signonUidText_ = new TextField("ИД пользователя", "JAVA", 10,
R                                                    TextField.ANY);
R     // Временный TBD
R     private TextField signonPwdText_ = new TextField("Пароль", "JTEAM1", 10,
```

```
R                                                   TextField.PASSWORD);
R     private TextField signonServerText_ = new TextField("Сервер ME Server", "localhost",
R                                                     10, TextField.ANY);
R     private StringItem signonStatusText_ = new StringItem("Состояние", NOT_SIGNED_ON);
R
R     // Поля TextField для формы CommandCall.
R     // TBD: максимальный размер; TBD: Текстовое описание???
R     private TextField cmdText_ = new TextField("Команда", "CRTLIB FRED", 256,
R                                           TextField.ANY);
R     private StringItem cmdMsgText_ = new StringItem("Сообщения", null);
R     private StringItem cmdStatusText_ = new StringItem("Состояние", null);
R
R     // Поля TextField для формы ProgramCall.
R     private StringItem pgmMsgDescription_ = new StringItem("Сообщения", null);
R     private StringItem pgmMsgText_ = new StringItem("Сообщения", null);
R
R     // Поля TextField для формы DataQueue.
R     private TextField dqInputText_ = new TextField("Данные для записи",
R                                         "Здравствуйте", 30, TextField.ANY);
R     private StringItem dqOutputText_ = new StringItem("Содержимое очереди данных",
R                                                     null);
R     private ChoiceGroup dqReadOrWrite_ = new ChoiceGroup("Действие",
R                                                       Choice.EXCLUSIVE,
R                                                       new String[] { DQ_WRITE, DQ_READ},
R                                                       null);
R     private StringItem dqStatusText_ = new StringItem("Состояние", null);
R
R
R     /**
R      * Создание нового ToolboxMidpDemo.
R      **/
R     public ToolboxMidpDemo()
R     {
R         display_ = Display.getDisplay(this);
R         // Примечание: В примере с KVM главная панель была создана с помощью TabbedPane.
R         // MIDP не содержит похожих классов, поэтому будет использован List.
R     }
R
R     /**
R      * Показ главного меню.
R      * Реализация абстрактного метода класса Midlet.
R      **/
R     protected void startApp()
R     {
R         main_.append(SIGN_ON, null);
R         main_.append(COMMAND_CALL, null);
R         main_.append(PROGRAM_CALL, null);
R         main_.append(DATA_QUEUE, null);
R         main_.append(ABOUT, null);
R
R         main_.addCommand(actionExit_);
R         main_.setCommandListener(this);
R
R         display_.setCurrent(main_);
R     }
R
R     // Реализация метода интерфейса CommandListener.
R     public void commandAction(Command action, Displayable dsp)
R     {
R         // Обработка всех действий 'exit' и 'back' одинакова.
R         if (action == actionExit_)
R         {
R             destroyApp(false);
R
R             notifyDestroyed();
R         }
R         else if (action == actionBack_)
```

```
R         {
R             // Возврат в главное меню.
R             display_.setCurrent(main_);
R         }
R         else if (dsp instanceof List)
R         {
R             List current = (List)dsp;
R
R             // Действие произошло на главной странице
R             if (current == main_)
R             {
R                 int   idx = current.getSelectedIndex();
R
R                 switch (idx)
R                 {
R                 case 0:     // SignOn
R                     showSignonForm();
R                     break;
R                 case 1:     // CommandCall
R                     showCmdForm();
R                     break;
R                 case 2:     // ProgramCall
R                     showPgmForm();
R                     break;
R                 case 3:     // DataQueue
R                     showDqForm();
R                     break;
R                 case 4:     // About
R                     showAboutForm();
R                     break;
R                 default:    // Ни одно из вышеперечисленных
R                     feedback("Внутренняя ошибка: необработанные элементы индекса в main: " + idx,
R                              AlertType.ERROR);
R                     break;
R                 }
R             } // current == main
R             else
R                 feedback("Внутренняя ошибка: просматриваемый объект имеет тип
R                           List, но не относится к main_.",
R                          AlertType.ERROR);
R         } // instanceof List
R         else if (dsp instanceof Form)
R         {
R             Form current = (Form)dsp;
R
R             if (current == signonForm_)
R             {
R                 if (action == actionSignon_)
R                 {
R                     // Создание системного объекта ToolboxME.
R                     system_ = new AS400(signonSystemText_.getString(),
R                                         signonUidText_.getString(),
R                                         signonPwdText_.getString(),
R                                         signonServerText_.getString());
R
R                     try
R                     {
R                         // Подключение к серверу.
R                         system_.connect();
R
R                         // Настройка текста о состоянии входа в систему.
R                         signonStatusText_.setText("Вход в систему выполнен.");
R
R                         // Показать окно подтверждения входа в систему.
R                         feedback("Регистрация в системе успешно выполнена.", AlertType.INFO, main_);
R
R                         // Заменять кнопку SignOn на кнопку SignOff.
```

```
R                       signonForm_.removeCommand(actionSignon_);
R                       signonForm_.addCommand(actionSignoff_);
R
R                       // Обновить индикатор.
R                       ticker_.setString("... Вошел в систему '" +
R                                         signonSystemText_.getString() + "' как '" +
R                                         signonUidText_.getString() + "' с помощью '" +
R                                         signonServerText_.getString() + "' ... ");
R                   }
R                   catch (Exception e)
R                   {
R                       e.printStackTrace();
R
R                       // Настройка текста о состоянии входа в систему.
R                       signonStatusText_.setText(NOT_SIGNED_ON);
R
R                       feedback("Регистрация в системе не выполнена.  " + e.getMessage(), AlertType.ERROR);
R                   }
R               }
R               else if (action == actionSignoff_)
R               {
R                   if (system_ == null)
R                       feedback("Внутренняя ошибка: пустая система.", AlertType.ERROR);
R                   else
R                   {
R                       try
R                       {
R                           // Disconnect from the server.
R                           system_.disconnect();
R                           system_ = null;
R
R                           // Настройка текста о состоянии входа в систему.
R                           signonStatusText_.setText(NOT_SIGNED_ON);
R
R                           // Показать окно подтверждения выхода из системы.
R                           feedback("Сеанс работы успешно завершен.", AlertType.INFO, main_);
R
R                           // Заменить кнопку SignOff на кнопку SignOn.
R                           signonForm_.removeCommand(actionSignoff_);
R                           signonForm_.addCommand(actionSignon_);
R
R                           // Обновить индикатор.
R                           ticker_.setString(NOT_SIGNED_ON);
R                       }
R                       catch (Exception e)
R                       {
R                           feedback(e.toString(), AlertType.ERROR);
R
R                           e.printStackTrace();
R
R                           signonStatusText_.setText("Ошибка.");
R
R                           feedback("Ошибка при завершении сеанса работы.", AlertType.ERROR);
R                       }
R                   }
R               }
R               else  // Ни одно из вышеперечисленных.
R               {
R                   feedback("Внутренняя ошибка: неизвестное действие.", AlertType.INFO);
R               }
R           } // signonForm_
R           else if (current == cmdcallForm_)
R           {
R               if (action == actionRun_)
R               {
R                   // Если пользователь не вошел в систему, выдать предупреждение.
R                   if (system_ == null)
```

```
R                   {
R                       feedback(NOT_SIGNED_ON, AlertType.ERROR);
R                       return;
R                   }
R
R                   // Получить команду, введенную пользователем на беспроводном устройстве.
R                   String cmdString = cmdText_.getString();
R
R                   // Если команда не задана, выдать предупреждение.
R                   if (cmdString == null || cmdString.length() == 0)
R                       feedback("Укажите команду.", AlertType.ERROR);
R                   else
R                   {
R                       try
R                       {
R                           // Запуск команды.
R                           String[] messages = CommandCall.run(system_, cmdString);
R
R                           StringBuffer status = new StringBuffer("Команда выполнена со значениями ");
R
R                           // Проверить, нет ли сообщений
R                           if (messages.length == 0)
R                           {
R                               status.append("ответных сообщений нет.");
R
R                               cmdMsgText_.setText(null);
R
R                               cmdStatusText_.setText("Команда успешно выполнена.");
R                           }
R                           else
R                           {
R                               if (messages.length == 1)
R                                   status.append("Получено одно сообщение.");
R                               else
R                                   status.append(messages.length + " сообщений получено.");
R
R                               // Если есть сообщения, показать только первое.
R                               cmdMsgText_.setText(messages[0]);
R
R                               cmdStatusText_.setText(status.toString());
R                           }
R
R                           repaint();
R                       }
R                       catch (Exception e)
R                       {
R                           feedback(e.toString(), AlertType.ERROR);
R
R                           e.printStackTrace();
R
R                           feedback("Ошибка при выполнении команды.", AlertType.ERROR);
R                       }
R                   }
R               }
R               else if (action == actionClear_)
R               {
R                   // Стереть текст команды и сообщения.
R                   cmdText_.setString("");
R
R                   cmdMsgText_.setText(null);
R
R                   cmdStatusText_.setText(null);
R
R                   repaint();
R               }
R               else  // Ни одно из вышеперечисленных.
R               {
```

```
R                       feedback("Внутренняя ошибка: неизвестное действие.", AlertType.INFO);
R                   }
R               } // cmdcallForm_
R               else if (current == pgmcallForm_)
R               {
R                   if (action == actionRun_)
R                   {
R                       // Если пользователь не вошел в систему перед вызовом программы, выдать предупреждение.
R                       if (system_ == null)
R                       {
R                           feedback(NOT_SIGNED_ON, AlertType.ERROR);
R                           return;
R                       }
R
R                       pgmMsgText_.setText(null);
R
R                       // Обратитесь к примеру PCML в документации к IBM Toolbox for Java.
R                       String pcmlName = "qsyrusri.pcml"; // Файл PCML, который следует использовать.
R                       String apiName = "qsyrusri";
R
R                       // Создать хэш-таблицу с входными параметрами для вызова программы.
R                       Hashtable parmsToSet = new Hashtable(2);
R                       parmsToSet.put("qsyrusri.receiverLength", "2048");
R                       parmsToSet.put("qsyrusri.profileName", signonUidText_.getString().toUpperCase());
R
R                       // Создать строковый массив для получаемых выходных параметров.
R                       String[] parmsToGet = { "qsyrusri.receiver.userProfile",
R                                               "qsyrusri.receiver.previousSignonDate",
R                                               "qsyrusri.receiver.previousSignonTime",
R                                               "qsyrusri.receiver.daysUntilPasswordExpires"};
R
R                       // Строковый массив с описаниями выдаваемых параметров.
R                       String[] displayParm = { "Профайл",
R                                                "Дата последнего входа в систему",
R                                                "Время последнего входа в систему",
R                                                "Срок действия пароля истек (дней)"};
R
R                       try
R                       {
R                           // Запуск программы.
R                           String[] valuesToGet = ProgramCall.run(system_,
R                                                                  pcmlName,
R                                                                  apiName,
R                                                                  parmsToSet,
R                                                                  parmsToGet);
R
R                           // Создать StringBuffer и занести в него все полученные параметры.
R                           StringBuffer txt = new StringBuffer();
R                           txt.append(displayParm[0] + ": " + valuesToGet[0] + "\n");
R
R                           char[] c = valuesToGet[1].toCharArray();
R                           txt.append(displayParm[1] + ": " + c[3] + c[4] + "/" +
R                                      c[5] + c[6] + "/" + c[1] + c[2] + "\n");
R
R                           char[] d = valuesToGet[2].toCharArray();
R                           txt.append(displayParm[2] + ": " + d[0] + d[1] + ":" + d[2] + d[3] + "\n");
R                           txt.append(displayParm[3] + ": " + valuesToGet[3] + "\n");
R
R                           // Задать отображаемый текст для результатов вызова программы.
R                           pgmMsgText_.setText(txt.toString());
R
R                           StringBuffer status = new StringBuffer("Программа выполнена со значениями ");
R
R                           if (valuesToGet.length == 0)
R                           {
R                               status.append("значения не получены.");
R
```

```
R                           feedback(status.toString(), AlertType.INFO);
R                       }
R                       else
R                       {
R                           if (valuesToGet.length == 1)
R                               status.append("Получено одно значение.");
R                           else
R                               status.append(valuesToGet.length + " значений получено.");
R
R                           feedback(status.toString(), AlertType.INFO);
R                       }
R                   }
R                   catch (Exception e)
R                   {
R                       feedback(e.toString(), AlertType.ERROR);
R
R                       e.printStackTrace();
R
R                       feedback("Ошибка при выполнении программы.", AlertType.ERROR);
R                   }
R               }
R               else if (action == actionClear_)
R               {
R                   // Очистка результатов вызова программы.
R                   pgmMsgText_.setText(null);
R
R                   repaint();
R               }
R           }  // pgmcallForm_
R           else if (current == dataqueueForm_)  // DataQueue
R           {
R               if (action == actionGo_)
R               {
R                   // Если перед выполнением операций с очередями данных пользователь не вошел в систему,
R                   // показать предупреждающее сообщение.
R                   if (system_ == null)
R                   {
R                       feedback(NOT_SIGNED_ON, AlertType.ERROR);
R
R                       return;
R                   }
R
R                   // Создать библиотеку для очереди данных.
R                   try
R                   {
R                       CommandCall.run(system_, "CRTLIB FRED");
R                   }
R                   catch (Exception e)
R                   {
R                   }
R
R                   // Выполнить команду создания очереди данных.
R                   try
R                   {
R                       CommandCall.run(system_, "CRTDTAQ FRED/MYDTAQ MAXLEN(2000)");
R                   }
R                   catch (Exception e)
R                   {
R                       feedback("Ошибка при создании очереди данных.  " + e.getMessage(),
R                                AlertType.WARNING);
R                   }
R
R                   try
R                   {
R                       // Определение выбранного действия (Чтение или Запись).
R                       if (dqReadOrWrite_.getString(dqReadOrWrite_.getSelectedIndex()).equals(DQ_WRITE))
R                       {
```

```
R                           // Запись
R                           dqOutputText_.setText(null);
R
R                           // Внести текстовые входные данные беспроводного устройства в
R                           // очередь сообщений.
R                           if (dqInputText_.getString().length() == 0)
R                               dqStatusText_.setText("Данные не заданы.");
R                           else
R                           {
R                               // Запись в очередь данных.
R                               DataQueue.write(system_,
R                                               "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ",
R                                               dqInputText_.getString().getBytes() );
R
R                               dqInputText_.setString(null);
R
R                               // Показать состояние.
R                               dqStatusText_.setText("Запись выполнена успешно.");
R                           }
R                       }
R                       else  // Чтение
R                       {
R                           // Чтение из очереди данных.
R                           byte[] b = DataQueue.readBytes(system_, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");
R
R                           // Определить, содержала ли очередь сообщений данные, и
R                           // вывести соответствующее сообщение.
R                           if (b == null)
R                           {
R                               dqStatusText_.setText("Нет доступных записей очереди данных.");
R
R                               dqOutputText_.setText(null);
R                           }
R                           else if (b.length == 0)
R                           {
R                               dqStatusText_.setText("Запись очереди данных пуста.");
R
R                               dqOutputText_.setText(null);
R                           }
R                           else
R                           {
R                               dqStatusText_.setText("Чтение выполнено успешно.");
R
R                               dqOutputText_.setText(new String(b));
R                           }
R                       }
R
R                       repaint();
R                   }
R                   catch (Exception e)
R                   {
R                       e.printStackTrace();
R
R                       feedback(e.toString(), AlertType.ERROR);
R
R                       feedback("Ошибка при выполнении команды.  " + e.getMessage(), AlertType.ERROR);
R                   }
R               }  // actionGo_
R               else if (action == actionClear_)
R               {
R                   // Очистка формы очереди данных.
R                   dqInputText_.setString("");
R
R                   dqOutputText_.setText(null);
R
R                   dqReadOrWrite_.setSelectedFlags(new boolean[] { true, false});
R
```

```
R                       dqStatusText_.setText(null);
R
R                       repaint();
R                   }
R                   else  // Ни одно из вышеперечисленных.
R                   {
R                       feedback("Внутренняя ошибка: неизвестное действие.", AlertType.INFO);
R                   }
R               } // dataqueueForm_
R               else if (current == aboutForm_)  // "О программе".
R               {
R                   // Для выполнения этой части кода необходимо воспользоваться кнопкой "Назад".
R               }  // Ни одно из вышеперечисленных.
R               else
R                   feedback("Внутренняя ошибка: неизвестная форма.", AlertType.ERROR);
R           } // instanceof Form
R           else
R               feedback("Внутренняя ошибка: неизвестный просматриваемый объект.", AlertType.ERROR);
R       }
R
R
R
R       /**
R        *  Вывод формы "О программе".
R        **/
R       private void showAboutForm()
R       {
R           // Если форма О программе пуста, создать и добавить ее.
R           if (aboutForm_ == null)
R           {
R               aboutForm_ = new Form(ABOUT);
R               aboutForm_.append(new StringItem(null,
R                                 "Это пример приложения MIDP, в котором применяется " +
R                                 "IBM Toolbox for Java Micro Edition (ToolboxME)."));
R
R               aboutForm_.addCommand(actionBack_);
R               aboutForm_.setCommandListener(this);
R           }
R
R           display_.setCurrent(aboutForm_);
R       }
R
R
R
R       /**
R        *  Вывод формы "Вход в систему".
R        **/
R       private void showSignonForm()
R       {
R           // Создание формы входа в систему.
R           if (signonForm_ == null)
R           {
R               signonForm_ = new Form(SIGN_ON);
R               signonForm_.append(signonSystemText_);
R               signonForm_.append(signonUidText_);
R               signonForm_.append(signonPwdText_);
R               signonForm_.append(signonServerText_);
R               signonForm_.append(signonStatusText_);
R               signonForm_.addCommand(actionBack_);
R               signonForm_.addCommand(actionSignon_);
R               signonForm_.setCommandListener(this);
R               signonForm_.setTicker(ticker_);
R           }
R
R           display_.setCurrent(signonForm_);
R       }
R
```

```
R
R
R       /**
R        *  Вывод формы "Вызов команды".
R        **/
R       private void showCmdForm()
R       {
R           // Создание формы вызова команды.
R           if (cmdcallForm_ == null)
R           {
R               cmdcallForm_ = new Form(COMMAND_CALL);
R               cmdcallForm_.append(cmdText_);
R               cmdcallForm_.append(cmdMsgText_);
R               cmdcallForm_.append(cmdStatusText_);
R               cmdcallForm_.addCommand(actionBack_);
R               cmdcallForm_.addCommand(actionClear_);
R               cmdcallForm_.addCommand(actionRun_);
R               cmdcallForm_.setCommandListener(this);
R               cmdcallForm_.setTicker(ticker_);
R           }
R
R           display_.setCurrent(cmdcallForm_);
R       }
R
R
R
R       /**
R        *  Вывод формы "Вызов программы".
R        **/
R       private void showPgmForm()
R       {
R           // Создание формы вызова программы.
R           if (pgmcallForm_ == null)
R           {
R               pgmcallForm_ = new Form(PROGRAM_CALL);
R               pgmcallForm_.append(new StringItem(null,
R                                                  "Будет вызван API Получить информацию о пользователе (QSYRUSRI) " +
R                                                  "API, после чего будут получены сведения о текущем " +
R                                                  "пользовательском профайле."));
R               pgmcallForm_.append(pgmMsgText_);
R               pgmcallForm_.addCommand(actionBack_);
R               pgmcallForm_.addCommand(actionClear_);
R               pgmcallForm_.addCommand(actionRun_);
R               pgmcallForm_.setCommandListener(this);
R               pgmcallForm_.setTicker(ticker_);
R           }
R
R           display_.setCurrent(pgmcallForm_);
R       }
R
R
R       /**
R        *  Вывод формы "Очередь данных".
R        **/
R       private void showDqForm()
R       {
R           // Создание формы очереди данных.
R           if (dataqueueForm_ == null)
R           {
R               dataqueueForm_ = new Form(DATA_QUEUE);
R               dataqueueForm_.append(dqInputText_);
R               dataqueueForm_.append(dqOutputText_);
R               dataqueueForm_.append(dqReadOrWrite_);
R               dataqueueForm_.append(dqStatusText_);
R               dataqueueForm_.addCommand(actionBack_);
R               dataqueueForm_.addCommand(actionClear_);
R               dataqueueForm_.addCommand(actionGo_);
```

```
R             dataqueueForm_.setCommandListener(this);
R             dataqueueForm_.setTicker(ticker_);
R         }
R
R         display_.setCurrent(dataqueueForm_);
R     }
R
R
R     private void feedback(String text, AlertType type)
R     {
R         feedback(text, type, display_.getCurrent());
R     }
R
R     /**
R      *  Этот метод применяется для создания окна диалога и вывода
R      *  уведомления с помощью отправки пользователю предупреждающего сообщения.
R      **/
R     private void feedback(String text, AlertType type, Displayable returnToForm)
R     {
R         System.err.flush();
R         System.out.flush();
R
R         Alert alert = new Alert("Alert", text, null, type);
R
R         if (type == AlertType.INFO)
R             alert.setTimeout(3000);  // время в миллисекундах
R         else
R             alert.setTimeout(Alert.FOREVER);  // Предупреждение аннулируется только пользователем.
R
R         display_.setCurrent(alert, returnToForm);
R     }
R
R
R     // Принудительная перерисовка текущей формы.
R     private void repaint()
R     {
R         Alert alert = new Alert("Обновление меню...", null, null, AlertType.INFO);
R         alert.setTimeout(1000);  // время в миллисекундах
R
R         display_.setCurrent(alert, display_.getCurrent());
R     }
R
R
R
R     /**
R      * Пауза, освобождение ненужного сейчас пространства.
R      * Реализация абстрактного метода класса Midlet.
R      **/
R     protected void pauseApp()
R     {
R         display_.setCurrent(null);
R     }
R
R
R
R     /**
R      * Общая очистка.
R      * Реализация абстрактного метода класса Midlet.
R      **/
R     protected void destroyApp(boolean unconditional)
R     {
R         // Disconnect from the server if the Midlet is being destroyed or exited.
R         if (system_ != null)
R         {
R             try
R             {
R                 system_.disconnect();
```

```
R             }
R             catch (Exception e)
R             {
R             }
R         }
R     }
R }
```

Информация, связанная с данной

“Профайл мобильных устройств (MIDP)” на стр. 354

Примеры: Классы Utility

This topic lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java utility classes.

Класс AS/400ToolboxJarMaker

- Пример: Распаковка класса AS400.class и зависимых классов из архива jt400.jar
- Пример: Разбиение файла jt400.jar на файлы объемом 300 Кб
- Пример: Удаление неиспользуемых файлов из архивного файла .jar
- Пример: Создание файла .jar сокращенного на 400 Кб объема путем пропуска таблиц преобразования с помощью параметра -ccsid

Класс CommandPrompter

- Пример: Применение CommandPrompter для вывода приглашения и запуска команды

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Применение CommandPrompter

This example program uses the CommandPrompter, CommandCall, and AS400Message classes to prompt for a command, run the command, and display any messages returned if the command does not run.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример применения CommandPrompter. Используя CommandPrompter, CommandCall и AS400Message,
// данная программа выдает приглашение команды, запускает команду и показывает
// все сообщения, возвращенные в случае, если команду выполнить не удалось.
//
// Формат вызова:
//    Prompter текст_команды
//
////////////////////////////////////////////////////////////////////////////

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];

  // Запуск CommandPrompter
  CommandPrompter cp = new CommandPrompter(frame, system, cmdName);
  if (cp.showDialog() == CommandPrompter.OK)
    {
    String cmdString = cp.getCommandString();
    System.out.println("Командная строка: " + cmdString);

    // Запуск команды, созданной в Prompter.
    CommandCall cmd = new CommandCall(system, cmdString);
    if (!cmd.run())
      {
      AS400Message[] msgList = cmd.getMessageList();
      for (int i = 0; i < msgList.length; ++i)
        {
          System.out.println(msgList[i].getText());
        }
      }
    }
  System.exit(0);
  }
}
```

Примеры: Классы Vaccess

This topic lists the code examples that are provided throughout the documentation of the IBM Toolbox for Java vaccess classes.

Класс AS400Panes

- Пример: Создание панели AS400DetailsPane для вывода списка пользователей, заданного в systemAS400DetailsPane
- Пример: Загрузка содержимого панели сведений до добавления ее в главное окно
- Пример: Применение AS400ListPane для вывода списка пользователей
- Пример: Использование AS400DetailsPane для вывода сообщений, полученных при вызове команды
- Пример: Использование AS400TreePane для вывода дерева каталогов
- Пример: Применение AS400ExplorerPane для просмотра ресурсов печати

Вызов команд

- Пример: Создание CommandCallButton
- R • Example: Adding the ActionCompletedListener to process all System i5 messages that a command generates
- Пример: Использование CommandCallMenuItem

Очереди данных

- Пример: Создание DataQueueDocument
- Пример: Использование DataQueueDocument

События при ошибках

- Пример: Обработка событий, связанных с ошибками
- Пример: Определение обработчика ошибок
- Пример: Использование собственного обработчика ошибок

Интегрированная файловая система

- Пример: Применение класса IFSFileDialog
- Пример: Применение класса IFSFileSystemView
- Пример: Применение класса IFSTextFileDocument

JDBC

- Пример: Применение драйвера JDBC для создания и заполнения таблицы
- Пример: Применение драйвера JDBC для обработки запроса к таблице и вывода ее содержимого
- Пример: Создание класса AS400JDBCDataSourcePane

Задания

- Пример: Создание VJobList и вывод списка с помощью AS400ExplorerPane
- Пример: Вывод списка заданий в панели проводника

Сообщения

- Пример: Применение класса VMessageQueue

Вызов программ

- Пример: Создание ProgramCallMenuItem
R • Example: Processing all program generated System i5 messages
- Пример: Добавление двух параметров
- Пример: Применение ProgramCallButton в приложении

Печать

- Пример: Применение класса VPrinter
- Пример: Класс VPrinterOutput

Доступ на уровне записей

- Пример: Создание объекта RecordListTablePane для просмотра записей с ключом, меньшим или равным указанному значению.
- Пример Применение класса RecordListFormPane

SpooledFileViewer

- Example: Creating a Spooled File Viewer to view a spooled file previously created on the system

SQL

- Пример: Применение класса SQLQueryBuilderPane
- Пример: Применение класса SQLResultSetTablePane

Системные значения

- Пример: Создание графического интерфейса для работы с системными значениями с помощью панели AS400Explorer

Пользователи и группы

- Пример: Создание VUserList с помощью AS400DetailsPane
- Пример: Использование AS400ListPane для создания списка пользователей с возможностью выбора

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> **Отказ от гарантий на предоставляемый код**
>
> IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.
>
> All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. По этой причине, IBM не может гарантировать их надежность и пригодность.
>
> Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие предполагаемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Пример: Применение класса VUserList

This example program presents a list of users on a system in a list pane, and allows selection of one or more users.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта VUserList. Эта программа показывает
// список пользователей системы и позволяет выбрать одного
// или несколько пользователей.
//
// Формат вызова:
//    VUserListExample система
//
////////////////////////////////////////////////////////////////////////

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)
    {
        // Если система не указана - вывод справочной информации
        // и завершение работы.
        if (args.length != 1)
        {
            System.out.println("Применение: VUserListExample система");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы указано
// первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);
```

```
            // Создание объекта VUserList. Этот объект представляет
// список пользователей, который будет показан на панели.
            VUserList userList = new VUserList (system);

            // Создание фрейма.
            JFrame f = new JFrame ("Пример объекта VUserList");

            // Создание адаптера окна ошибок.
// В этом окне пользователю будет показана информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели для вывода списка пользователей.
            // Получение информации от сервера методом load.
            listPane = new AS400ListPane (userList);
            listPane.addErrorListener (errorHandler);
            listPane.load ();

            // При закрытии фрейма - вывод списка выбранных пользователей и выход.
f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    reportSelectedUsers ();
                    System.exit (0);
                }
            });

            // Размещение нового фрейма.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("По центру", listPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


    private static void reportSelectedUsers ()
    {
        VObject[] selectedUsers = listPane.getSelectedObjects ();

        if (selectedUsers.length == 0)
            System.out.println ("Пользователи не выбраны.");
        else
        {
            System.out.println ("Были выбраны следующие пользователи:");
            for (int i = 0; i < selectedUsers.length; ++i)
                System.out.println (selectedUsers[i]);
        }
    }


}
```

Пример: Применение класса VMessageList

This program presents a detailed view of messages returned from a command call.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////////
//
// VMessageList example.  This program presents a detailed
// подробный текст сообщений, полученных при обработке команды.
//
// Формат вызова:
//    VMessageListExample система
//
// Пример применения класса VMessageList IBM Toolbox for 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)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
        if (args.length != 1)
        {
            System.out.println("Применение: VMessageListExample system");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание объекта CommandCall для запуска команды.
            CommandCall command = new CommandCall (system);
            command.run ("CRTLIB FRED");

            // Создание объекта VMessageList с сообщениями,
            // полученными после вызова команды.
            VMessageList messageList = new VMessageList (command.getMessageList ());

            // Создание фрейма.
            JFrame f = new JFrame ("Пример объекта VMessageList");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели для вывода списка сообщений.
            // Загрузка информации методом load.
            AS400DetailsPane detailsPane = new AS400DetailsPane (messageList);
            detailsPane.addErrorListener (errorHandler);
            detailsPane.load ();

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });
```

```
            // Размещение фрейма и панели с подробной информацией.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", detailsPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Пример: Применение класса VIFSDirectory

This example presents a tree view of some directories in the IFS.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта VIFSDirectory. Данная программа
// показывает иерархию каталогов интегрированной файловой системы.
//
// Формат вызова:
//    VIFSDirectoryExample система
//
// Пример применения класса VIFSDirectory IBM Toolbox for 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)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
        if (args.length != 1)
        {
            System.out.println("Применение: VIFSDirectoryExample система");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание объекта VIFSDirectory,
            // представляющего корень дерева каталогов.
            VIFSDirectory directory = new VIFSDirectory (system, "/QIBM/ProdData");

            // Создание фрейма.
            JFrame f = new JFrame ("Пример применения VIFSDirectory");
```

```
            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели для вывода иерархии каталогов.
            // Загрузка информации из системы.
            AS400TreePane treePane = new AS400TreePane (directory);
            treePane.addErrorListener (errorHandler);
            treePane.load ();

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("По центру", treePane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Пример: Применение класса VPrinters

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта VPrinters. Эта программа показывает
// сетевые ресурсы печати.
//
// Формат вызова:
//    VPrintersExample система
//
// Пример применения класса VPrinters IBM Toolbox for 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)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
```

```
        if (args.length != 1)
        {
            System.out.println("Применение: VPrintersExample система");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание объекта VPrinters, представляющего
            // список принтеров, подключенных к системе.
VPrinters printers = new VPrinters (system);

            // Создание фрейма.
            JFrame f = new JFrame ("Пример применения класса VPrinters");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели проводника для просмотра информации о
            // Загрузка информации из системы методом load.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (printers);
            explorerPane.addErrorListener (errorHandler);
              explorerPane.load ();

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма с панелью проводника.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", explorerPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Пример: CommandCallMenuItem

This IBM Toolbox for Java example program demonstrates how to use a menu item that calls a server command. It will display any messages that are returned in a dialog.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта CommandCallMenuItemExample. Эта программа
// демонстрирует применение пункта меню, вызывающего команду сервера.
// В окне диалога будут показаны все сообщения, полученные
```

```
// в результате выполнения команды.
//
// Формат вызова:
//    CommandCallMenuItemExample система
//
////////////////////////////////////////////////////////////////////////

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)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
        if (args.length != 1)
        {
            System.out.println("Применение:  CommandCallMenuItemExample система");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание фрейма.
            f = new JFrame ("Пример пункта меню вызова команды"

            // Создание окна диалога ошибок.  В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание объекта CommandCallMenuItem для запуска команды.
            CommandCallMenuItem menuItem =
               new CommandCallMenuItem ("Clear library FRED", null, system, "CLRLIB FRED");
            menuItem.addErrorListener (errorHandler);

            // Обработчик события завершения команды
            // покажет все полученные сообщения в окне диалога.
            menuItem.addActionCompletedListener (new ActionCompletedListener ()
            {
                public void actionCompleted (ActionCompletedEvent event)
                {
                    // Получение списка сообщений из источника событий
                    CommandCallMenuItem item = (CommandCallMenuItem) event.getSource ();
                    AS400Message[] messageList = item.getMessageList ();

                    // Вывод сообщений с помощью панели AS400DetailsPane
                    VMessageList vmessageList = new VMessageList (messageList);
                    AS400DetailsPane messageDetails = new AS400DetailsPane (vmessageList);
                    messageDetails.load ();

                    // Размещение панели в окне диалога.
                    JDialog dialog = new JDialog(f);
```

```
                dialog.getContentPane().setLayout(new BorderLayout());
                dialog.getContentPane().add("Center"messageDetails);
                dialog.pack();
                dialog.setVisible(true);
            }
        });

        // Создание меню с одним элементом
        JMenu menu = new JMenu ("Вызовы команд сервера");
        menu.add (menuItem);

        JMenuBar menuBar = new JMenuBar ();
        menuBar.add (menu);

        f.getRootPane ().setJMenuBar (menuBar);

        // Завершение работы программы в случае закрытия фрейма пользователем.
        f.addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent event)
            {
                System.exit (0);
            }
        });

        // Размещение фрейма и панели с подробной информацией.
        f.getContentPane ().setLayout (new BorderLayout ());
        f.setSize (300, 400);
        f.show ();
      }
      catch (Exception e)
      {
         System.out.println ("Ошибка: " + e.getMessage ());
         System.exit (0);
      }
   }


}
```

Пример: Работа с классом DataQueueDocument

This program demonstrates how to use a document that is associated with a server data queue.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////
//
// Пример применения объекта DataQueueDocument. Эта программа
// демонстрирует работу с документом, связанным с очередью данных сервера.
//
// Формат вызова:
//    DataQueueDocumentExample система 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)
    {
        // Если не указана система или аргумент read|write -
        // вывод справки и завершение работы.
        if (args.length != 2)
        {
            System.out.println("Применение:  чтение|запись системы в класс DataQueueDocumentExample");
            return;
        }

        rw = args[1].equalsIgnoreCase ("чтение");
        String mode = rw ? "Чтение" : "Запись";

        try
        {
            // Создание двух фреймов.
            JFrame f =
               new JFrame ("Пример документа очереди данных - " + mode);

            // Создание окна диалога ошибок.  В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание адаптера курсора. Он будет изменять положение
// курсора при чтении или записи данных в очередь.
            WorkingCursorAdapter cursorAdapter = new WorkingCursorAdapter (f);

            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание полного имени очереди данных.
            QSYSObjectPathName dqName = new QSYSObjectPathName ("QGPL", "JAVATALK", "DTAQ");

            // Проверка наличия очереди данных.
            DataQueue dq = new DataQueue (system, dqName.getPath ());
            try
            {
                dq.create (200);
            }
            catch (Exception e)
            {
                // Исключительные ситуации игнорируются. Предполагается,
// что очередь данных уже существует.
            }

            // Создание объекта DataQueueDocument.
            dqDocument = new DataQueueDocument (system, dqName.getPath ());
            dqDocument.addErrorListener (errorHandler);
            dqDocument.addWorkingListener (cursorAdapter);

            // Создание текстового поля для представления документа.
            text = new JTextField (dqDocument, "", 40);
            text.setEditable (! rw);
// Для работы программы необходимо знать, выполняется
            // чтение или запись. Для этого предусмотрена
// следующая кнопка.
            Button button = new Button (mode);
            button.addActionListener (new ActionListener ()
            {
                public void actionPerformed (ActionEvent event)
                {
                    if (rw)
                        dqDocument.read ();
                    else {
                        dqDocument.write ();
```

```
                    text.setText ("");
                }
            }
        });

        // При закрытии фрейма - завершение работы.
        f.addWindowListener (new WindowAdapter ()
        {
            public void windowClosing (WindowEvent event)
            {
                System.exit (0);
            }
        });

        // Размещение фрейма.
        f.getContentPane ().setLayout (new FlowLayout ());
        f.getContentPane ().add (text);
        f.getContentPane ().add (button);
        f.pack ();
        f.show ();
      }
      catch (Exception e)
      {
         System.out.println ("Ошибка: " + e.getMessage ());
         System.exit (0);
      }
   }
}
```

Пример: Применение IFSFileDialog

This example illustrates the use of the IBM Toolbox for Java IFSFileDialog class.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта FileDialog.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.awt.*;
import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;

public class FileDialogExample extends Object
{

   public static void main(String[] parameters)
   {

      System.out.println( " " );



      // Если имя системы не задано - вывод справки и завершение работы.

      if (parameters.length >= 1)
      {

         // Первый параметр - имя системы, содержащей файлы.

         String system           = parameters[0];
```

```
        try
        {
            // Создание объекта AS400, представляющего систему, содержащую файлы.
            // Подключение к файловому серверу. После подключения будет показано
// входа в систему.

            AS400 as400 = new AS400(system);
            as400.connectService(AS400.FILE);



            // Создание фрейма для окна диалога.

            Frame frame = new Frame();



            // Создание объекта, представляющего окно выбора файла.

            IFSFileDialog fileDialog = new IFSFileDialog(frame, "File Open", as400);



            // Создание списка фильтров и добавление фильтров
            // в окно диалога.

            FileFilter[] filterList = {
               new FileFilter("Все файлы (*.*)", "*.*"),
               new FileFilter("Исполняемые файлы (*.exe)", "*.exe"),
               new FileFilter("Файлы HTML (*.html)", "*.html"),
               new FileFilter("Изображения (*.gif)", "*.gif"),
               new FileFilter("Текстовые файлы (*.txt)", "*.txt")};

            fileDialog.setFileFilter(filterList, 0);



            // Указание текста для кнопки "OK" окна диалога.

            fileDialog.setOkButtonText("Открыть");



            // Указание текста для кнопки "Отмена" окна диалога.

            fileDialog.setCancelButtonText("Отмена");



            // Настройка начального каталога для окна диалога.

            fileDialog.setDirectory("/");



            // Вывод окна. Ожидание нажатия пользователем кнопки OK или Cancel

            int pressed = fileDialog.showDialog();



            // При нажатии кнопки OK - считать полное имя
            // выбранного файла.

            if (pressed == IFSFileDialog.OK)
            {
               System.out.println("Выбор пользователя: " +
```

```
                                fileDialog.getAbsolutePath());
            }



            // При нажатии кнопки Cancel - вывод сообщения.

            else if (pressed == IFSFileDialog.CANCEL)
            {
               System.out.println("Пользователь отменил операцию");
            }

            else
               System.out.println("Пользователь не нажал кнопки Открыть или Отмена");

         }
         catch(Exception e)
         {
            // Если при выполнении какой-либо операции возник сбой -
            // появляется сообщение об ошибке и возникает исключительная ситуация.

            System.out.println("Сбой операции в окне диалога");
            System.out.println(e);
         }
      }


      // Если заданы неверные параметры, вывести текст справки.

      else
      {
         System.out.println("");
         System.out.println("");
         System.out.println("");
         System.out.println("Параметры указаны неверно. Формат команды:");
System.out.println("");
         System.out.println("  FileDialogExample система");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println("  system = System i5");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println("");
         System.out.println("  FileDialogExample mySystem");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}
```

Пример: Применение класса IFSTextFileDocument

This program demonstrates how to use a document that is associated with a text file in the integrated file system.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////
//
// Пример применения объекта IFSTextFileDocument. Эта программа
// use a document that is associated with a text file in the
// файлом из интегрированной файловой системы.
//
// Формат вызова:
```

```
//    IFSTextFileDocumentExample система полное-имя-файла
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class IFSTextFileDocumentExample
{



    private static IFSTextFileDocument  document;
    private static JTextPane            text;



    public static void main(String[] args)
    {
        // Если были указаны не все параметры -
        // вывод справки и завершение работы.
        if (args.length != 2)
        {
            System.out.println("Применение: Полный путь класса IFSTextFileDocumentExample в системе");
            return;
        }

        try
        {
            // Создание двух фреймов.
            JFrame f = new JFrame ("Пример документа текстового файла IFS");

            // Создание окна ошибок.  В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание адаптера курсора. Он будет изменять положение
// курсора при чтении и записи данных в файл.
            WorkingCursorAdapter cursorAdapter = new WorkingCursorAdapter (f);

            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание и загрузка документа, связанного с текстовым файлом IFS.
            document = new IFSTextFileDocument (system, args[1]);
            document.addErrorListener (errorHandler);
            document.addWorkingListener (cursorAdapter);
            document.load ();

            // Создание текстового поля для представления документа.
            text = new JTextPane (document);
            text.setSize (new Dimension (500, 500));

            // Создание полосы прокрутки, используемой совместно с текстовым полем.
            JScrollPane scroll = new JScrollPane (text);
            scroll.setHorizontalScrollBarPolicy (JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
            scroll.setVerticalScrollBarPolicy (JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);

            // Создание строки меню с одним элементом.
            MenuBar menuBar = new MenuBar ();
            Menu menu = new Menu ("Файл");
            menuBar.add (menu);
```

```
            // Добавление элементов меню Загрузить и Сохранить.
            MenuItem load = new MenuItem ("Загрузить");
            load.addActionListener (new ActionListener ()
            {
                public void actionPerformed (ActionEvent event)
                {
                    document.load ();
                }
            });
            menu.add (load);

            MenuItem save = new MenuItem ("Сохранить");
            save.addActionListener (new ActionListener ()
            {
                public void actionPerformed (ActionEvent event)
                {
                    document.save ();
                }
            });
            menu.add (save);

            // При закрытии фрейма - завершение работы.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", scroll);
            f.setMenuBar (menuBar);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Класс AS400JDBCDataSourcePane

The AS400JDBCDataSourcePane class presents the property values of an AS400JDBCDataSource object. В объект AS400JDBCDataSource могут быть внесены дополнительные изменения.

Класс AS400JDBCDataSourcePane является расширением класса JComponent. Для просмотра свойств источника данных с помощью класса AS400JDBCDataSourcePane укажите источник данных в конструкторе класса или вызовите метод setDataSource() после создания объекта AS400JDBCDataSourcePane. Для применения изменений, внесенных в графический пользовательский интерфейс (GUI) источника данных, вызовите метод applyChanges().

Пример: Применение объекта AS400JDBCDataSourcePane

В приведенном ниже примере создается объект AS400JDBCDataSourcePane и кнопка **OK**, которые затем добавляются во фрейм. Изменения, внесенные в GUI, применяются к источнику данных после нажатия кнопки **OK**.

```
        // Создание источника данных.
        myDataSource = new AS400JDBCDataSource();

        // Создание окна панели и кнопки OK.
        JFrame frame = new JFrame ("JDBC Data Source Properties");

        // Создание панели источника данных.
        dataSourcePane = new AS400JDBCDataSourcePane(myDataSource);

        // Создание кнопки OK
        JButton okButton = new JButton("OK");

        // Добавление объекта ActionListener для кнопки OK. При нажатии кнопки
// OK будет вызван метод applyChanges() для применения изменений
        // к источнику данных.
        okButton.addActionListener(new ActionListener()
           {
              public void actionPerformed(ActionEvent ev)
              {
                 // Применить все изменения, внесенные на панели источника данных
                 // к источнику данных.  Если все изменения будут применены
// успешно, получить источник данных на основе панели.
                 if (dataSourcePane.applyChanges())
                 {
                    System.out.println("Нажата кнопка OK");
                    myDataSource = dataSourcePane.getDataSource();
                    System.out.println(myDataSource.getServerName());
                 }
              }
           }
        );

        // Настройка фрейма для показа панели и кнопки OK.
        frame.getContentPane().setLayout(new BorderLayout() );
        frame.getContentPane ().add ("Center", dataSourcePane);
        frame.getContentPane ().add ("South", okButton);

        // Упаковка фрейма.
        frame.pack ();

        // Показать панель и кнопку OK.
        frame.show ();
```

Информация, связанная с данной

AS400JDBCDataSourcePane Javadoc

Пример: Применение класса VJobList для вывода списка заданий

This program presents a list of jobs in an explorer pane.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////
//
// Пример применения списка заданий. В этом примере список заданий
// отображается в окне проводника.
//
// Формат вызова:
//    VJobListExample система
//
// Пример применения класса AS400ExplorerPane IBM Toolbox for 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)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
        if (args.length != 1)
        {
            System.out.println("Применение: VJobListExample system");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание объекта VJobList, представляющего список
            // заданий с именем QZDASOINIT.
            VJobList jobList = new VJobList (system);
            jobList.setName ("QZDASOINIT");

            // Создание фрейма.
            JFrame f = new JFrame ("Пример списка заданий");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели проводника для просмотра списка заданий.
            // Загрузка информации из системы методом load.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (jobList);
            explorerPane.addErrorListener (errorHandler);
            explorerPane.load ();

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма с панелью проводника.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", explorerPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Пример: Применение класса VMessageQueue

This program presents a message queue in an explorer pane.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример очереди вывода. В этом примере очередь вывода
// отображается в окне проводника.
//
// Формат вызова:
//    VMessageQueueExample система
//
// Пример применения класса VMessageQueue IBM Toolbox for Java.
//
////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VMessageQueueExample
{


    public static void main(String[] args)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
        if (args.length != 1)
        {
            System.out.println("Применение: VMessageQueueExample система");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Запуск процедуры входа в систему для определения имени пользователя.
            system.connectService (AS400.COMMAND);

            // Создание объекта VMessageQueue, представляющего
            // очередь сообщений текущего пользователя.
            VMessageQueue queue = new VMessageQueue (system,
                QSYSObjectPathName.toPath ("QUSRSYS", system.getUserId (),
                "MSGQ"));

            // Создание фрейма.
            JFrame f = new JFrame ("Пример очереди вывода");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели для вывода очереди сообщений.
            // Загрузка информации из системы методом load.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (queue);
            explorerPane.addErrorListener (errorHandler);
            explorerPane.load ();

            // Завершение работы программы в случае закрытия фрейма пользователем.
```

```
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма с панелью проводника.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", explorerPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Пример: Создание кнопки для вызова программы на сервере

This program demonstrates how to use a button that calls a program on the server. It will exchange data with the server program via an input and output parameter.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта ProgramCallButton. В этой программе
// создается кнопка для вызова программы на сервере. Для обмена
// данными с программой сервера применяются параметры ввода-вывода.
//
// Формат вызова:
//    ProgramCallButtonExample система
//
// Пример применения класса ProgramCallButton IBM Toolbox for 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;


    // Создание объекта ProgramCallButtonExample, затем вызов
    // нестатической версии main(). В статической версии
// переменные класса (parm1, parm2, ...) должны быть объявлены
// статическими. Это запрещает их использование в обработчике
// события в Java версий 1.1.7 и 1.1.8.
    public static void main(String[] args)
    {
```

```
       ProgramCallButtonExample me = new ProgramCallButtonExample();
       me.Main(args);
    }

    public void Main (String[] args)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
        if (args.length != 1)
        {
            System.out.println("Применение: ProgramCallButtonExample система");
            return;
        }

        try
        {
            // Создание фрейма.
            JFrame f = new JFrame ("Пример применения кнопки вызова программы");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Настройка пути к программе.
            QSYSObjectPathName programName = new QSYSObjectPathName ("QSYS",
                "QWCRSSTS", "PGM");

            // Создание объекта ProgramCallButton.  На кнопке
            // будет написано "Обновить", значка не будет.
            ProgramCallButton button = new ProgramCallButton ("Обновить", null);
            button.setSystem (system);
            button.setProgram (programName.getPath ());
            button.addErrorListener (errorHandler);

            // Первый параметр - выходной параметр размером 64 байта.
            parm1 = new ProgramParameter (64);
            button.addParameter (parm1);

            // Второй параметр служит для настройки размера буфера
            // первого параметра. Его значение всегда будет равно
// 64. Значение 64 необходимо преобразовать
// из формата int в формат AS/400.
            AS400Bin4 parm2Converter = new AS400Bin4 ();
            byte[] parm2Bytes = parm2Converter.toBytes (64);
            parm2 = new ProgramParameter (parm2Bytes);
            button.addParameter (parm2);

            // Третий параметр задает формат информации о состоянии. Его значение
// будет всегда равно SSTS0200. Это строковое значение, которое
            // также должно быть преобразовано в формат сервера.
            AS400Text parm3Converter = new AS400Text (8, system);
            byte[] parm3Bytes = parm3Converter.toBytes ("SSTS0200");
            parm3 = new ProgramParameter (parm3Bytes);
            button.addParameter (parm3);

            // Четвертый параметр предназначен для сброса данных статистики.
            // В качестве десятисимвольной строки будет всегда передаваться значение *NO.
            AS400Text parm4Converter = new AS400Text (10, system);
            byte[] parm4Bytes = parm4Converter.toBytes ("*NO       ");
            parm4 = new ProgramParameter (parm4Bytes);
            button.addParameter (parm4);

            // Пятый параметр предназначен для информации об ошибках. Он
```

```
// является параметром ввода-вывода. В данном примере этот
// параметр не применяется, но его значение необходимо задать,
            // так как в противном случае будет передано неверное
            // число параметров.
            byte[] parm5Bytes = new byte[32];
            parm5 = new ProgramParameter (parm5Bytes, 0);
            button.addParameter (parm5);

            // Программа возвращает определенный объем данных.
            // Эту информацию необходимо предоставить пользователю.
            // В данном случае для этого применяются простые метки
            // и текстовые поля.
            JLabel cpuLabel = new JLabel ("Использование CPU: ");
            cpuField = new JTextField (10);
            cpuField.setEditable (false);

            JLabel dasdLabel = new JLabel ("Использование DASD: ");
            dasdField = new JTextField (10);
            dasdField.setEditable (false);

            JLabel jobsLabel = new JLabel ("Число активных заданий: ");
            jobsField = new JTextField (10);
            jobsField.setEditable (false);

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // При вызове программы необходимо обрабатывать
            // информацию, возвращаемую в первом параметре.
            // Формат данных этого параметра описан в документации
            // по вызываемой программе.
            button.addActionCompletedListener (new ActionCompletedListener ()
            {
                public void actionCompleted (ActionCompletedEvent event)
                {
                    try
                    {
                       // Получение данных из первого параметра.
                       // Эти данные находятся в формате сервера.
                       byte[] parm1Bytes = parm1.getOutputData ();

                       // Все необходимые нам значения имеют тип
                       // int. Для всех значений можно создать
// один объект преобразования.
                       AS400Bin4 parm1Converter = new AS400Bin4 ();

                       // Получение значения использования CPU начиная с байта 32.
                       // Запись его в соответствующее текстовое поле.
                       int cpu = parm1Converter.toInt (parm1Bytes, 32);
                       cpuField.setText (Integer.toString (cpu / 10) + "%");

                       // Получение значения использования DASD начиная с байта 52.
                       // Запись его в соответствующее текстовое поле.
                       int dasd = parm1Converter.toInt (parm1Bytes, 52);
                       dasdField.setText (Integer.toString (dasd / 10000) + "%");

                       // Получение числа активных заданий начиная с байта 36.
                       // Запись его в соответствующее текстовое поле.
                       int jobs = parm1Converter.toInt (parm1Bytes, 36);
                       jobsField.setText (Integer.toString (jobs));
                    }
```

```
                catch (Exception e) { e.printStackTrace(); }
            }
        });

        // Размещение фрейма.
        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 ("В центре", outputPanel);
        f.getContentPane ().add ("Внизу", buttonPanel);
        f.pack ();
        f.show ();
      }
      catch (Exception e)
      {
         System.out.println ("Ошибка: " + e.getMessage ());
         System.exit (0);
      }
   }


}
```

Пример: Работа с классом VPrinter

This example program presents a printer and its spooled files in an explorer pane.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример VPrinter. Данная программа показывает информацию о принтере
// и список буферных файлов в панели проводника.
//
// Формат вызова:
//    VPrinterExample имя-системы
//
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrinterExample
{


    public static void main(String[] args)
    {

        // Если пользователь не укажет имя принтера, то будет показана
        // информация о принтере OS2VPRT;
        String printerName = "OS2VPRT";

        // Если не указана система, то будет выдана справочная
```

```
        // информация и работа будет завершена.
        if (args.length == 0)
        {
            System.out.println("Формат:  VPrinterExample имя-системы имя-принтера");
            return;
        }


        // Если указано имя принтера, то его следует применять вместо значения по умолчанию
        if (args.length > 1)
           printerName = args[1];


        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);


            // Создание объекта Printer (из пакета access),
            // представляющего принтер, и объекта VPrinter
            // для вывода информации о буферных файлах.
            Printer   printer = new Printer(system, printerName);
            VPrinter vprinter = new VPrinter(printer);

            // Создание нового фрейма.
            JFrame f = new JFrame ("Пример VPrinter");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);


            // Создание панели проводника для просмотра информации
            // о принтере и списка буферных файлов. Загрузка информации.
AS400ExplorerPane explorerPane = new AS400ExplorerPane (vprinter);
            explorerPane.addErrorListener (errorHandler);
                    explorerPane.load ();


            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма с панелью проводника.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", explorerPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }
}
```

Пример: Применение класса VPrinters

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////
//
// Пример применения объекта VPrinters. Эта программа показывает
// сетевые ресурсы печати.
//
// Формат вызова:
//    VPrintersExample система
//
// Пример применения класса VPrinters IBM Toolbox for 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)
    {
        // Если не указана система, то будет выдана справочная
        // информация и работа будет завершена.
        if (args.length != 1)
        {
            System.out.println("Применение: VPrintersExample система");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);

            // Создание объекта VPrinters, представляющего
            // список принтеров, подключенных к системе.
VPrinters printers = new VPrinters (system);

            // Создание фрейма.
            JFrame f = new JFrame ("Пример применения класса VPrinters");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание панели проводника для просмотра информации о
            // Загрузка информации из системы методом load.
            AS400ExplorerPane explorerPane = new AS400ExplorerPane (printers);
            explorerPane.addErrorListener (errorHandler);
              explorerPane.load ();

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });
```

```
            // Размещение фрейма с панелью проводника.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", explorerPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Пример VPrinterOutput

This example program presents a list of spooled files on the server. All spooled files, or only spooled files for a specific user, can be displayed.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример VPrinterOutput. Данная программа показывает список
// буферных файлов сервера. Пользователь может просмотреть
// все файлы или файлы конкретного пользователя.
//
// Формат вызова:
//    VPrinterOutputExample system <user>
//
//    (Если пользователь не задан, то будут показаны все буферные файлы
//    системы. Внимание - вывод списка всех буферных файлов системы
//    может занять длительное время.)
//
////////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VPrinterOutputExample
{


    public static void main(String[] args)
    {

        // Если не указана система, то будет выдана справочная информация и работа будет завершена.
        if (args.length == 0)
        {
            System.out.println("Usage:  VPrinterOutputExample system <user>");
            return;
        }

        try
        {
            // Создание объекта AS400. Имя системы задается
            // первым параметром в командной строке.
            AS400 system = new AS400 (args[0]);
            system.connectService(AS400.PRINT);

            // Создание объекта VPrinterOutput.
```

```
            VPrinterOutput printerOutput = new VPrinterOutput(system);

            // Если в командной строке был указан пользователь, то
            // ИД пользователя заносится в объект printerObject.
            if (args.length > 1)
               printerOutput.setUserFilter(args[1]);


            // Создание нового фрейма.
            JFrame f = new JFrame ("Пример VPrinterOutput");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);


            // Создание панели для вывода списка буферных файлов.
            // Загрузка информации из системы методом load.
            AS400DetailsPane detailsPane = new AS400DetailsPane (printerOutput);
            detailsPane.addErrorListener (errorHandler);
            detailsPane.load ();


            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма и панели с подробной информацией.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Center", detailsPane);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }
}
```

Пример: Применение класса SQLQueryBuilderPane

This program presents a query builder that allows the user to build a SQL query.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта SQLQueryBuilderPane. Эта программа
// that allows the user to build a SQL query.
//
// Формат вызова:
//    SQLQueryBuilderPaneExample система
//
// Пример применения классов SQLQueryBuilderPane
// и SQLResultSetFormPane IBM Toolbox for Java.
//
///////////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class SQLQueryBuilderPaneExample
{


    // Соединение, общее для всех компонентов.
    private SQLConnection connection;



    // Обработчик ошибок, общий для всех компонентов.
    private ErrorDialogAdapter errorHandler;



    // Панель конструктора запросов.
    private SQLQueryBuilderPane queryBuilderPane;



    // Функция main языка Java. В этой функции создается экземпляр собственного
// класса данной программы и вызывается метод Main() этого экземпляра.
// Это позволяет обойти ограничения на работу статических методов
// с динамическими данными, особенно расположенными во внутренних классах.
    // Для упрощения программы число статических переменных и методов сведено
// к минимуму.
    public static void main(String[] args)
    {
       SQLQueryBuilderPaneExample me = new SQLQueryBuilderPaneExample();
       me.Main(args);
    }


    public void Main (String[] args)
    {
        // Если система не указана -
        // вывод справки и завершение работы.
        if (args.length != 1)
        {
            System.out.println("Применение: SQLQueryBuilderPaneExample система");
            return;
        }

        try
        {
            // Регистрация драйвера JDBC IBM Toolbox for Java.
            DriverManager.registerDriver(new AS400JDBCDriver());

            // Создание объекта SQLConnection. Имя системы задается
            // первым параметром в командной строке.
            connection = new SQLConnection ("jdbc:as400://" + args[0]);

            // Создание фрейма.
            JFrame f = new JFrame ("Пример применения класса SQLQueryBuilderPane");

            // Создание окна диалога ошибок. В нем пользователю
            // будет выдаваться информация об ошибках.
            errorHandler = new ErrorDialogAdapter (f);

            // Создание панели конструктора запросов SQL для
            // создания запроса. Загрузка данных системы, необходимых
            // для применения конструктора запросов.
            queryBuilderPane = new SQLQueryBuilderPane (connection);
```

```
            queryBuilderPane.addErrorListener (errorHandler);
            queryBuilderPane.load ();

            // Создание кнопки для вывода результатов обработки запроса
            // в другом фрейме.
            JButton resultSetButton = new JButton ("Показать набор результатов");
            resultSetButton.addActionListener (new ActionListener ()
            {
                public void actionPerformed (ActionEvent event)
                {
                    showFormPane (queryBuilderPane.getQuery ());
                }
            });

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма в окне конструктора запросов.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("В центре", queryBuilderPane);
            f.getContentPane ().add ("Внизу", resultSetButton);
            f.pack ();
            f.show ();
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }



    private void showFormPane (String query)
    {
        // Создание нового фрейма для результата обработки запроса.
        JFrame f = new JFrame (query);

        // Создание панели для вывода результатов обработки запроса SQL.
        // Загрузка результатов из системы.
SQLResultSetFormPane formPane = new SQLResultSetFormPane (connection, query);
        formPane.addErrorListener (errorHandler);
        formPane.load ();

        // Размещение фрейма.
        f.getContentPane ().setLayout (new BorderLayout ());
        f.getContentPane ().add ("В центре", formPane);
        f.pack ();
        f.show ();
    }


}
```

Пример: Применение класса SQLResultSetTablePane

This program presents the contents of a table in a table pane. There is a SQLStatementDocument that allows the user to type in any SQL statement. In addition, there is a button that allows the user to delete all rows of the table.

Примечание: Ознакомьтесь с важной юридической информацией, приведенной в разделе Отказ от гарантий на предоставляемый код.

```
///////////////////////////////////////////////////////////////////////
//
// Пример применения объекта SQLResultSetTablePane. Эта программа
// a table in a table pane.  There is a SQLStatementDocument that allows
// операторы SQL с помощью класса SQLStatementDocument. Кроме того,
// that allows the user to delete all rows of the table.
//
// Формат вызова:
//    SQLResultSetTablePaneExample система таблица
//
// Пример применения классов SQLQueryBuilderPane, SQLResultSetFormPane и
// SQLStatementButton IBM Toolbox for Java.
//
///////////////////////////////////////////////////////////////////////

import com.ibm.as400.access.*;
import com.ibm.as400.vaccess.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

public class SQLResultSetTablePaneExample
{



    private static SQLStatementDocument     document;
    private static SQLResultSetTablePane    tablePane;



    public static void main(String[] args)
    {
        // Если система не указана -
        // вывод справки и завершение работы.
        if (args.length != 2)
        {
            System.out.println("Применение: SQLResultSetTablePaneExample система таблица");
            return;
        }

        try
        {
            // Регистрация драйвера JDBC IBM Toolbox for Java.
            DriverManager.registerDriver(new AS400JDBCDriver());

            // Создание объекта SQLConnection. Имя системы задается
            // первым параметром в командной строке.              // Соединение применяется всеми компонентами.
            SQLConnection connection = new SQLConnection ("jdbc:as400://" + args[0]);

            // Создание фрейма.
            JFrame f = new JFrame ("Пример применения класса SQLResultSetTablePane");

            // Создание окна диалога ошибок. В нем пользователю
            // В этом окне будет выдаваться информация об ошибках.
// Обработчик ошибок применяется всеми компонентами.
            ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (f);

            // Создание объекта SQLStatementDocument,
            // позволяющего пользователю ввести запрос.
            document = new SQLStatementDocument (connection, "");
            document.addErrorListener (errorHandler);

            // Создание текстового поля.
            JTextField textField = new JTextField (document,
                "Введите в этом поле оператор SQL.", 50);
```

```
            // Создание кнопки, удаляющей все строки таблицы.
            SQLStatementButton deleteAllButton = new SQLStatementButton ("Удалить все строки");
            deleteAllButton.setConnection (connection);
            deleteAllButton.setSQLStatement ("УДАЛИТЬ ИЗ " + args[1]);
            deleteAllButton.addErrorListener (errorHandler);

            // Создание панели для вывода результатов обработки запроса SQL.
            // Загрузка данных.
tablePane = new SQLResultSetTablePane (connection, "ВЫБЕРИТЕ * ИЗ " + args[1]);
            tablePane.addErrorListener (errorHandler);
            tablePane.load ();

            // После нажатия в текстовом поле клавиши Enter -
            // выполнение оператора SQL и обновление таблицы.
            textField.addKeyListener (new KeyAdapter ()
            {
                public void keyPressed (KeyEvent event)
                {
                    if (event.getKeyCode () == KeyEvent.VK_ENTER)
                    {
                        // Оператор SELECT обрабатывается панелью таблицы,
// остальные операторы - объектом document.
                        String sql = document.getSQLStatement ();
                        if (sql.toUpperCase ().startsWith ("ВЫБЕРИТЕ"))
                        {
                            try
                            {
                                tablePane.setQuery (sql);
                            }
                            catch (Exception e)
                            {
                                // Игнорировать.
                            }
                            tablePane.load ();
                        }
                        else
                            document.execute ();
                    }
                }
            });

            // После удаления всех строк с помощью кнопки - обновление таблицы.
deleteAllButton.addActionCompletedListener (new ActionCompletedListener ()
            {
                public void actionCompleted (ActionCompletedEvent event)
                {
                    tablePane.load ();
                }
            });

            // Завершение работы программы в случае закрытия фрейма пользователем.
            f.addWindowListener (new WindowAdapter ()
            {
                public void windowClosing (WindowEvent event)
                {
                    System.exit (0);
                }
            });

            // Размещение фрейма в окне конструктора запросов.
            f.getContentPane ().setLayout (new BorderLayout ());
            f.getContentPane ().add ("Вверху", textField);
            f.getContentPane ().add ("В центре", tablePane);
            f.getContentPane ().add ("Внизу", deleteAllButton);
            f.pack ();
            f.show ();
```

```
        }
        catch (Exception e)
        {
           System.out.println ("Ошибка: " + e.getMessage ());
           System.exit (0);
        }
    }


}
```

Примеры: Компонент XPCML

В этом разделе перечислены примеры программ, встречающиеся в документации по компонентам XPCML IBM Toolbox for Java.

- “Пример: Уплотнение существующего документа XPCML” на стр. 761
- “Пример: Уплотнение существующего документа XPCML с исходным кодом Java” на стр. 761
- “Пример: Создание объекта ProgramCallDocument с помощью уплотненного файла XPCML” на стр. 764
- “Пример: Получение результатов вызова программы в виде уплотненного файла XPCML” на стр. 764
- “Пример: Получение результатов вызова программы в виде файла XPCML”
- “Пример: Передача значений параметров в виде файла XPCML” на стр. 758
- “Пример: Передача массивов значений параметров в виде файла XPCML” на стр. 759
- “Пример: Преобразование документа PCML в документ XPCML” на стр. 443

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Пример: Получение результатов вызова программы в виде файла XPCML

The following example shows how you can construct an XPCML ProgramCallDocument, call a System i program, and retrieve the results of the program call as XPCML.

В примере используются следующие компоненты:

- Документ XPCML qgyolaus.xpcml, который содержит спецификации параметров и программы, а также входные значения
- Код Java, который создает объект ProgramCallDocument, применяет файл XPCML и вызывает программу QGYOLAUS
- Результаты вызова программы, которые код Java создает в виде документа XPCML и сохраняет в файле XPCMLOut.xpcml

Обратите внимание на то, как задаются массивы данных в исходном и созданном файлах XPCML. Выходной параметр - элемент qgyolaus.receiver - является объектом XPCML arrayOfStructParm с атрибутом, устанавливающим значение счетчика listInfo.rcdsReturned. Приведенный ниже пример содержит только часть вывода программы QGYOLAUS. If the example included all the output, the code might list 89 users under the <arrayOfStructParm> XPCML tag.

For arrays of structs, XPCML uses the <struct_i> XPCML tag to delimit each structParm element. Each <struct_i> tag indicates that the data enclosed within it is one element of type autu0150 struct. The index attribute of the <struct_i> tag specifies the element of the array for the struct.

For arrays of simple types, such as arrayOfStringParm, arrayOfIntParm, and so on, the <i> XPCML tag lists array elements.

Документ XPCML qgyolaus.xpcml

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<!-- XPCML source for calling "Open List of Authorized Users" -->
<!-- (QGYOLAUS) API -->

  <!-- Format AUTU0150 - Other formats are available -->
  <struct name="autu0150">
    <stringParm name="name" length="10"/>
    <stringParm name="userOrGroup" length="1"/>
    <stringParm name="groupMembers" length="1"/>
    <stringParm name="description" length="50"/>
  </struct>

  <!-- List information structure (common for "Open List" type APIs) -->
  <struct name="listInfo">
    <intParm name="totalRcds"/>
    <intParm name="rcdsReturned">0</rcdsReturned>
    <hexBinaryParm name="rqsHandle" totalBytes="4"/>
    <intParm name="rcdLength"/>
    <stringParm name="infoComplete" length="1"/>
    <stringParm name="dateCreated" length="7"/>
    <stringParm name="timeCreated" length="6"/>
    <stringParm name="listStatus" length="1"/>
    <hexBinaryParm totalBytes="1"/>
    <unsignedIntParm name="lengthOfInfo"/>
    <intParm name="firstRecord"/>
    <hexBinaryParm totalBytes="40"/>
  </struct>

  <!-- Program QGYOLAUS and its parameter list for retrieving -->
  <!-- AUTU0150 format -->

  <program name="QGYOLAUS" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm"
     parseOrder="listInfo receiver">
   <parameterList>
     // Выходные значения --- массив структур autu0150
     <arrayOfStructParm name="receiver" count="listInfo.rcdsReturned"
        passDirection="out" outputSize="receiverLength" struct="autu0150"/>
     // Входные значения
     <intParm name="receiverLength" passDirection="in">16384</intParm>
     <structParm name="listInfo" passDirection="out" struct="listInfo"/>
     // Входные значения
     <intParm name="rcdsToReturn" passDirection="in">264</intParm>
     <stringParm name="format" passDirection="in" length="10">
        AUTU0150</stringParm>
     <stringParm name="selection" passDirection="in" length="10">
        *USER</stringParm>
     <stringParm name="member" passDirection="in" length="10">
        *NONE</stringParm>
     <intParm name="errorCode" passDirection="in">0</intParm>
   </parameterList>
  </program>
```

Код Java, создающий объект ProgramCallDocument и вызывающий программу QGYOLAUS с помощью документа XPCML

```
system = new AS400();
// Создание объекта ProgramCallDocument, в котором будет выполняться анализ файла.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "QGYOLAUS.xpcml");

// Вызов программы QGYOLAUS
boolean rc = xpcmlDoc.callProgram("QGYOLAUS");

// Получение результатов программы в виде документа XPCML и сохранение
// их в файле XPCMLOut.xpcml
if (rc)     // Программа успешно выполнена
   xpcmlDoc.generateXPCML("QGYOLAUS","XPCMLOut.xpcml");
```

Результаты вызова программы в виде документа XPCML, сохраненные в файле XPCMLOut.xpcml

```
<?xml version="1.0" ?>
<xpcml version="4.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation='xpcml.xsd' >

  <program name="QGYOLAUS" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm"
     parseOrder="listInfo receiver">
  <parameterList>
      <arrayOfStructParm name="receiver" passDirection="out"
          count="listInfo.rcdsReturned" outputSize="receiverLength"
          struct="autu0150">
        <struct_i index="0">
          <stringParm name="name" length="10">JANEDOW</stringParm>
          <stringParm name="userOrGroup" length="1">0</stringParm>
          <stringParm name="groupMembers" length="1">0</stringParm>
          <stringParm name="description" length="50">
             Jane Doe</stringParm>
        </struct_i>
        <struct_i index="1">
          <stringParm name="name" length="10">BOBS</stringParm>
          <stringParm name="userOrGroup" length="1">0</stringParm>
          <stringParm name="groupMembers" length="1">0</stringParm>
          <stringParm name="description" length="50">
             Bob Smith</stringParm>
        </struct_i>

        <!-- Наличие дополнительных записей зависит от количества -->
        <!-- имен пользователей, возвращенных программой.-->
        <!-- В данном случае список содержит 89 имен пользователей.    -->

      </arrayOfStructParm>     <!-- End of user array -->
      <intParm name="receiverLength" passDirection="in">
         16384</intParm>
      <structParm name="listInfo" passDirection="out"
         struct="listInfo">
          <intParm name="totalRcds">89</intParm>
          <intParm name="rcdsReturned">89</intParm>
          <hexBinaryParm name="rqsHandle" totalBytes="4">
             00000001==</hexBinaryParm>
          <intParm name="rcdLength">62</intParm>
          <stringParm name="infoComplete" length="1">C</stringParm>
          <stringParm name="dateCreated" length="7">
             1030321</stringParm>
          <stringParm name="timeCreated" length="6">
             120927</stringParm>
          <stringParm name="listStatus" length="1">2</stringParm>
          <hexBinaryParm totalBytes="1"></hexBinaryParm>
          <unsignedIntParm name="lengthOfInfo">
```

```
            5518</unsignedIntParm>
          <intParm name="firstRecord">1</intParm>
      </structParm>
      <intParm name="rcdsToReturn" passDirection="in">264</intParm>
      <stringParm name="format" passDirection="in" length="10">
         AUTU0150</stringParm>
      <stringParm name="selection" passDirection="in" length="10">
         *USER</stringParm>
      <stringParm name="member" passDirection="in" length="10">
         *NONE</stringParm>
      <intParm name="errorCode" passDirection="in">0</intParm>
  </parameterList>
  </program>
</xpcml>
```

Пример: Передача значений параметров в виде файла XPCML

Значения параметров программы можно задать в исходном файле XPCML. При чтении и анализе файла XPCML для каждого параметра, значение которого задано в файле XPCML, автоматически вызывается метод setValue объекта ProgramCallDocument. Это позволяет пользователю не задавать значения сложных структур и массивов вручную в коде Java.

В приведенных ниже примерах в файле XPCML вызываются две разных программы, prog1 и prog2. В них используется входной параметр s1Ref. В первом примере при всех вызовах задаются разные значения s1Ref. Во втором примере при каждом вызове применяется одно и то же значение s1Ref, что демонстрирует способ задания постоянных значений входных параметров.

Пример: Передача разных значений входных параметров

В приведенном ниже примере после чтения и анализа документа с помощью анализатора XML значение элемента prog1.s1Ref.s2Ref.s2p1[0] будет равно prog1Val_1, а элемента prog1.s1Ref.s2Ref.s2p1[1] - prog1Val_2.

```
    <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<struct name="s1">
   <stringParm name="s1p1"/>
   <structParm name="s2Ref" struct="s2"/>
</struct>

<struct name="s2">
  <stringParm name="s2p1" length="10"/>
  <arrayOfStringParm name="parm1" count="2"/>
</struct>

<program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
 <parameterList>
   <structParm name="s1Ref" struct="s1" passDirection="in" >
     <stringParm name="s1p1">prog1Val</stringParm>
     <structParm name="s2Ref" struct="s2">
         <stringParm name="s2p1" length="10">prog1Val</stringParm>
         <arrayOfStringParm name="parm1" count="2">
            <i>prog1Val_1</i>
            <i>prog1Val_2</i>
         </arrayOfStringParm>
     </structParm>
   </structParm>
 </parameterList>
</program>

<program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
 <parameterList>
   <structParm name="s1Ref" struct="s1" passDirection="in" >
     <stringParm name="s1p1">prog2Val</stringParm>
     <structParm name="s2Ref" struct="s2">
         <stringParm name="s2p1" length="10">prog2Val</stringParm>
```

```
          <arrayOfStringParm name="parm1" count="2">
             <i>prog2Val_1</i>
             <i>prog2Val_2</i>
          </arrayOfStringParm>
       </structParm>
     </structParm>
   </parameterList>
  </program>
 </xpcml>
```

Пример: Передача одинаковых значений входных параметров

В приведенном ниже примере после чтения и анализа документа с помощью анализатора XML значение элемента prog1.s1Ref.s2Ref.s2p1[0] будет равно constantVal_1, а элемента prog1.s1Ref.s2Ref.s2p1[1] - constantVal_2.

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

 <struct name="s1" >
    <stringParm name="s1p1">constantVal</stringParm>
    <structParm name="s2Ref" struct="s2"/>
 </struct>

 <struct name="s2">
   <stringParm name="s2p1" length="10">constantVal</stringParm>
   <arrayOfStringParm name="parm1" count="2">
      <i>constantVal_1</i>
      <i>constantVal_2</i>
   </arrayOfStringParm>
 </struct>

 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" />
  </parameterList>
 </program>

 <program name="prog2" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" />
  </parameterList>
 </program>
</xpcml>
```

Пример: Передача массивов значений параметров в виде файла XPCML

В приведенном ниже примере показано, как можно передать массив значений параметров с помощью данных массива structParm и массива объектов struct.

При передаче данных массивов с помощью файлов XPCML необходимо использовать атрибут count:

- Задайте атрибут count для элемента массива
- Присвойте атрибуту count значение числа элементов, которое содержит массив на момент анализа документа

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

 <struct name="s1" >
    <stringParm name="s1p1"/>
    <struct name="s1Array">
      <stringParm name="s1Ap1"/>
    </struct>
 </struct>
```

```
 <struct name="s2">
    <stringParm name="s2p1"/>
 </struct>

 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
  <parameterList>
    <structParm name="s1Ref" struct="s1" passDirection="in" >
        <stringParm name="s1p1">Value 1</stringParm>
        <arrayOfStruct name="s1Array" count="2">
           <struct_i>
               <stringParm name="s1Ap1">Value 1</stringParm>
           </struct_i>
           <struct_i>
               <stringParm name="s1Ap1">Value 2</stringParm>
           </struct_i>
         </arrayOfStruct>
    </structParm>
    <arrayOfStructParm name="s2Ref" struct="s2" count="2" passDirection="in" >
      <struct_i>
          <stringParm name="s2p1">Value 1</stringParm>
      </struct_i>
      <struct_i>
          <stringParm name="s2p1">Value 2</stringParm>
      </struct_i>
    </arrayOfStructParm>
  </parameterList>
 </program>
</xpcml>
```

Например, в следующем файле XPCML создается массив из 3 элементов intParms, в котором первому элементу присваивается значение 12, второму - 100, третьему - 4:

```
<?xml version="1.0" ?>
<xpcml version="4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation='xpcml.xsd' >
  <program name="prog1" path="/QSYS.lib/MYLIG.lib/PROG1.pgm">
  <parameterList>
    <arrayOfIntParm name="intArray" count="3">
      <i>12</i>
      <i>100</i>
      <i>4</i>
    </arrayOfIntParm>
  </parameterList>
  </program>
</xpcml>
```

Using the index attribute of the <i> and <struct_i> tags to set array values

You can use the index attribute of the <i> and <struct_i> tags to help you set array values. В приведенном ниже примере в файле XPCML первому элементу массива присваивается значение 4, второму - 100, третьему - 12.

```
<?xml version="1.0" ?>
<xpcml version="4.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation='xpcml.xsd' >

  <program name="prog1" path="/QSYS.lib/MYLIG.lib/PROG1.pgm">
  <parameterList>
    <arrayOfIntParm name="intArray" count="3">
      <i index="2">12</i>
      <i index="1">100</i>
      <i index="0">4</i>
    </arrayOfIntParm>
  </parameterList>
  </program>
</xpcml>
```

Пример: Уплотнение существующего документа XPCML

Ниже приведен пример уплотнения существующего документа XPCML. Он содержит исходный документ XPCML, полученный уплотненный документ XPCML и расширенную схему.

Исходный документ XPCML

```
<?xml version="1.0" encoding="UTF-8"?>
 <xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">
  <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
    <parameterList>
       <stringParm name="parm1" passDirection="in" passMode="value"
             minvrm="V5R2M0" ccsid="37" length="10">Value 1</stringParm>
    </parameterList>
  </program>
 </xpcml>
```

Уплотненный документ XPCML

```
<?xml version="1.0" encoding="UTF-8"?>
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">
 <program name="prog1" path="/QSYS.LIB/W95LIB.LIB/PROG1.PGM">
    <parameterList>
       <parm1_>Value 1</parm1_>
    </parameterList>
 </program>
</xpcml>
```

Созданная схема

```
<!-- parm1's XSD definition -->
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
 <!-- Link back to XPCML.xsd -->
 <xs:include schemaLocation='xpcml.xsd'/>
  <xs:element name="parm1_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
     <xs:simpleContent>
     <xs:restriction base="stringParmType">
        <!-- Attributes defined for parm1 -->
        <xs:attribute name="name" type="string50" fixed="parm1" />
        <xs:attribute name="length" type="xs:string" fixed="10" />
        <xs:attribute name="passMode" type="xs:string" fixed="value" />
        <xs:attribute name="ccsid" type="xs:string" fixed="37" />
        <xs:attribute name="minvrm" type="xs:string" fixed="V5R2M0" />
     </xs:restriction>
     </xs:simpleContent>
   </xs:complexType>
  </xs:element>
</schema>
```

Пример: Уплотнение существующего документа XPCML с исходным кодом Java

Ниже приведен пример уплотнения существующего документа XPCML. The example includes original XPCML source, the resulting condensed XPCML, the Java code that calls condenseXPCML(), and a few of the newly generated type definitions in the extended schema.

Исходный документ XPCML

```
<!-- Fully specified XPCML source -->
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<struct name="qualifiedJobName">
   <stringParm name="jobName" length="10">*</stringParm>
   <stringParm name="userName" length="10"/>
```

```
   <stringParm name="jobNumber" length="6"/>
</struct>

<struct name="jobi0100">
   <intParm name="numberOfBytesReturned"/>
   <intParm name="numberOfBytesAvailable"/>
   <structParm name="qualifiedJobName" struct="qualifiedJobName"/>
   <hexBinaryParm name="internalJobIdentifier" totalBytes="16"/>
   <stringParm name="jobStatus" length="10"/>
   <stringParm name="jobType" length="1"/>
   <stringParm name="jobSubtype" length="1"/>
   <stringParm length="2"/>
   <intParm name="runPriority"/>
   <intParm name="timeSlice"/>
   <intParm name="defaultWait"/>
   <stringParm name="purge" length="10"/>
</struct>

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
      <structParm name="receiverVariable" passDirection="out"
         outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
      <intParm name="lengthOfReceiverVariable" passDirection="in">86</intParm>
      <stringParm name="formatName" passDirection="in" length="8">JOBI0100</stringParm>
      <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
      <hexBinaryParm name="internalJobIdentifier"
         passDirection="in" totalBytes="16"> </hexBinaryParm>
      <intParm name="errorCode" passDirection="in">0</intParm>
   </parameterList>
</program>
</xpcml>
```

Код Java, выполняющий уплотнение исходного документа XPCML

```
try {
   FileInputStream fullStream = new FileInputStream("myXPCML.xpcml");
   FileOutputStream condensedStream = new FileOutputStream("myCondensedXPCML.xpcml");
   FileOutputStream xsdStream = new FileOutputStream("myXSD.xsd");
   ProgramCallDocument.condenseXPCML(fullStream, xsdStream, condensedStream, "myXSD.xsd");

}
catch (Exception e) {
  System.out.println("ошибка:  - "+e.getMessage());
  e.printStackTrace();
}
```

Уплотненный документ XPCML: myCondensedXPCML.xpcml

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">

<struct name="qualifiedJobName">
   <jobName_>*</jobName_>
   <userName_/>
   <jobNumber_/>
</struct>

<struct name="jobi0100">
   <numberOfBytesReturned_/>
   <numberOfBytesAvailable_/>
   <structParm name="qualifiedJobName" struct="qualifiedJobName"/>
   <internalJobIdentifier_/>
   <jobStatus_/>
   <jobType_/>
   <jobSubtype_/>
   <stringParm length="2"/>
   <runPriority_/>
   <timeSlice_/>
```

```
   <defaultWait_/>
   <purge_/>
</struct>

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
      <structParm name="receiverVariable" passDirection="out"
         outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
      <lengthOfReceiverVariable_>86</lengthOfReceiverVariable_>
      <formatName_>JOBI0100</formatName_>
      <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
      <internalJobIdentifier_> </internalJobIdentifier_>
      <errorCode_>0</errorCode_>
   </parameterList>
</program>
</xpcml>
```

Некоторые определения типов в созданной схеме: myXSD.xsd

```
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<xs:include schemaLocation='xpcml.xsd'/>

<xs:element name="jobName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="jobName" />
         <xs:attribute name="length" type="xs:string" fixed="10" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="userName_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="userName" />
         <xs:attribute name="length" type="xs:string" fixed="10" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="jobNumber_" substitutionGroup="stringParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="jobNumber" />
         <xs:attribute name="length" type="xs:string" fixed="6" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="lengthOfReceiverVariable_" substitutionGroup="intParmGroup" >
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="intParmType">
         <xs:attribute name="name" type="string50" fixed="lengthOfReceiverVariable" />
         <xs:attribute name="passDirection" type="passDirectionType" fixed="in" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<xs:element name="formatName_" substitutionGroup="stringParmGroup" >
```

```
   <xs:complexType>
      <xs:simpleContent>
      <xs:restriction base="stringParmType">
         <xs:attribute name="name" type="string50" fixed="formatName" />
         <xs:attribute name="length" type="xs:string" fixed="8" />
         <xs:attribute name="passDirection" type="passDirectionType" fixed="in" />
      </xs:restriction>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>

<!-- More type definitions for each newly defined type follow here -->
</xs:schema>
```

Пример: Создание объекта ProgramCallDocument с помощью уплотненного файла XPCML

Некоторые конструкторы ProgramCallDocument поддерживают работу с уплотненными исходными файлами XPCML и соответствующими схемами (файлами .xsd). Это позволяет создавать объекты ProgramCallDocument с помощью уплотненных документов XPCML.

При применении упомянутых выше конструкторов необходимо указывать следующие параметры:

- Текстовое имя уплотненного файла XPCML
- Объект InputStream, который содержит определения типов, созданные с помощью метода condenseXPCML()

При применении этих конструкторов загружаются и анализируются уплотненные файлы XPCML. Кроме того, при этом все ошибки анализатора заносятся в протокол. После завершения анализа конструктор создает объект ProgramCallDocument.

В приведенном ниже примере программы на Java с помощью уплотненного файла XPCML создается объект ProgramCallDocument. При запуске примера предполагается, что:

- Имя уплотненного файла XPCML - myCondensedXPCML.xpcml
- Имя расширенной схемы - myXSD.xsd

После создания объекта ProgramCallDocument с его помощью вызывается программа qusrjobi_jobi0100.

```
AS400 system = new AS400();
// Создание объекта ProgramCallDocument и анализ файла.
ProgramCallDocument xpcmlDoc =
  new ProgramCallDocument(system,
                          "myCondensedXPCML.xpcml",
                          new FileInputStream("myXSD.xsd"));
boolean rc = xpcmlDoc.callProgram("qusrjobi_jobi0100");
```

Примечание: Код XPCML, с помощью которого вызывается программа (после создания объекта ProgramCallDocument), не отличается от кода, применяемого в документах PCML.

Пример: Получение результатов вызова программы в виде уплотненного файла XPCML

You use the same process to obtain the results of a program call as condensed XPCML or noncondensed XPCML by calling ProgramCallDocument.generateXPCML().

Use setXsdName() to specify the name of the extended schema, which generateXPCML() uses to generate the noNamespaceSchemaLocation attribute of the <xpcml> tag in the condensed XPCML.

Если полученные результаты (в виде уплотненного XPCML) используются в качестве исходных данных для создания другого объекта ProgramCallDocument, необходимо применять метод setXsdName(). Необходимо задать имя файла расширенной схемы, которую будет применять анализатор.

Например, в следующем примере на основе результатов вызова программы создается уплотненный документ XPCML.

```
AS400 system = new AS400();

// Создание объекта ProgramCallDocument и анализ файла.
ProgramCallDocument xpcmlDoc =
   new ProgramCallDocument(system, "myCondensedXPCML.xpcml", new FileInputStream("myXSD.xsd"));

boolean rc = xpcmlDoc.callProgram("qusrjobi_jobi0100");

if (rc)     // Программа успешно выполнена
{
  xpcmlDoc.setXsdName("myXSD.xsd");
  xpcmlDoc.generateXPCML("qusrjobi_jobi0100","XPCMLOut.xpcml");
}
```

В приведенном ниже примере результаты вызова программы представлены в виде уплотненного документа XPCML:

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="myXSD.xsd" version="4.0">

<program name="qusrjobi_jobi0100" path="/QSYS.LIB/QUSRJOBI.PGM">
   <parameterList>
   <structParm name="receiverVariable" passDirection="out"
       outputSize="lengthOfReceiverVariable" struct="jobi0100"/>
            <numberOfBytesReturned_>100</numberOfBytesReturned_>
            <numberOfBytesAvailable_>100</numberOfBytesAvailable_>
            <structParm name="qualifiedJobName"
                struct="qualifiedJobName">
                <jobName_>*</jobName_>
                <userName_/>
                <jobNumber_/>
            </structParm>
            <internalJobIdentifier_/>
            <jobStatus_>ACTIVE</jobStatus_>
            <jobType_>PJ</jobType>
            <jobSubtype_/>
            <stringParm length="2"/>
            <runPriority_>5</runPriority>
            <timeSlice_/>
            <defaultWait_>10</defaultWait_>
            <purge_/>
        </structParm>
        <lengthOfReceiverVariable_>86</lengthOfReceiverVariable_>
        <formatName_>JOBI0100</formatName_>
        <structParm name="qualifiedJobName" passDirection="in" struct="qualifiedJobName"/>
        <internalJobIdentifier_> </internalJobIdentifier_>
        <errorCode_>0</errorCode_>
</parameterList>
</program>
</xpcml>
```

Related information for IBM Toolbox for Java

The following list includes Web sites and Information Center topics that relate to theIBM Toolbox for Java information.

IBM Toolbox for Java resources

Use the following sites to learn more about the IBM Toolbox for Java:

- IBM Toolbox for Java and JTOpen  : Offers information about service packs, performance tips, examples, and much more. You can also download a zipped package of this information, including the Javadocs.

- IBM Toolbox for Java Frequently Asked Questions (FAQ) : Provides answers to questions about performance, troubleshooting, JDBC, and more.
- IBM Toolbox for Java and JTOpen forum : Offers an effective way to communicate with the community of Java programmers who use IBM Toolbox for Java and the IBM Toolbox for Java developers themselves.

Ресурсы IBM Toolbox for Java 2 Micro Edition

Use the following sites to learn more about ToolboxME and the Java implementation of wireless technologies:

- IBM Toolbox for Java and JTOpen : Offers more information about ToolboxME.
- IBM alphaWorks Wireless : Информация о новых беспроводных технологиях, бесплатные ресурсы для загрузки и ссылки на ресурсы разработчиков.
- На странице Sun Java 2 Platform, Micro Edition приведена дополнительная информация о беспроводных технологиях Java, включая такие технологии, как:
  - Виртуальная машина K (KVM)
  - Конфигурация подключенных устройств с ограниченными ресурсами (CLDC)
  - Профайл мобильных устройств (MIDP)
- Java Wireless Developer : Offers a wide range of technical information for Java wireless application developers.
- Средства для создания таких приложений:
  - IBM WebSphere Studio Device Developer
  - Java 2 Platform Micro Edition, Wireless Toolkit

Java

Java - это язык программирования, предназначенный для разработки переносимых объектно-ориентированных приложений и аплетов. Дополнительная информация о Java приведена на следующих Web-сайтах:

- IBM developerWorks Java technology zone : информационные, обучающие и прикладные ресурсы, предназначенные для помощи в работе с Java, продуктами IBM и другими технологиями для создания бизнес-приложений.
- IBM alphaWorks Java : Информация о новых технологиях Java, ресурсы для загрузки и ссылки на ресурсы разработчиков.
- На странице ″The Source for Java Technology″ компании Sun Microsystems приведена информация о различных способах применения Java, включая новые технологии.

Java Naming and Directory Interface

- Java Naming and Directory Interface (JNDI) : Offers an overview of JNDI, technical information, examples, and a list of available service providers.
- Directory Server (LDAP) : Provides information about LDAP (Lightweight Directory Access Protocol) on i5/OS.

Java Secure Socket Extension

- Страница Java Secure Socket Extension (JSSE) содержит общую информацию о JSSE и ссылки на другие ресурсы.

Сервлеты

Сервлетами называются небольшие программы на Java, выполняющиеся на сервере и обрабатывающие запросы одного или нескольких клиентов (у каждого из которых запущен браузер) к одной или нескольким базам данных. Так как сервлеты являются программами на Java, запросы могут обрабатываться несколькими нитями одного процесса, что существенно экономит ресурсы системы. Дополнительная информация о сервлетах приведена на следующих Web-сайтах:

- Страница IBM Websphere, IBM PartnerWorld содержит информацию о Web-сервере приложений на основе сервлетов.
- Java Servlet technology : Техническая информация, инструкции по работе с сервлетами и перечень полезных средств.

XHTML

Язык XHTML можно рассматривать как следующую версию языка HTML 4.0. Он сочетает в себе достоинства HTML 4.0 и XML (в частности, расширяемость). Дополнительная информация о XHTML приведена на следующих Web-сайтах:

- The Web Developer's Virtual Library : Обзор XHTML, включающий примеры и ссылки на дополнительную информацию.
- На странице W3C приведены технические сведения о стандартах XHTML и различные рекомендации.

XML

Расширяемый язык описаний (XML) - это метаязык, позволяющий создать структурное описание информации, понятное как человеку, так и компьютеру. Метаязык позволяет определить язык описания документа и его структуру. Дополнительная информация о XML приведена на следующих Web-сайтах:

- IBM developerWorks XML zone : Информация о разработках компании IBM в области XML и их применении в электронной коммерции
- IBM alphaWorks XML : Информация о новых стандартах и средствах XML, а также файлы для загрузки и ссылки на ресурсы разработчиков.
- Страница W3C XML содержит список технических ресурсов для разработчиков XML.
- XML.com : Информация о роли XML в современной компьютерной индустрии.
- XML.org : Новости и информация о пользователях и разработчиках XML, в том числе новости отрасли, календари событий и другая информация.
- XML Cover Pages : Полный электронный справочник по XML, SGML и другим стандартам, связанным с XML, таким как XSL и XSLT.

Другие ссылки

- IBM HTTP Server for i5/OS : Provides information, resources, and tips about the IBM HTTP Server for i5/OS.
- L System i Access for Windows : Offers information about System i Access for Windows, including downloads, FAQs, and links to additional sites.
- IBM WebSphere Host On-Demand : Provides information about the browser-based emulator that offers support for S/390, i5/OS, and DEC/Unix emulation.

- IBM Support and downloads  : Портал, посвященный поддержке аппаратного и программного обеспечения IBM.

Лицензия на исходный код и отказ от обязательств

IBM предоставляет вам неисключительную лицензию на использование всех примеров программного кода. Разрешается создавать на их основе программный код, необходимый вам.

ПРИ УСЛОВИИ СОБЛЮДЕНИЯ ВСЕХ НЕ ДОПУСКАЮЩИХ ИСКЛЮЧЕНИЙ ГАРАНТИЙ, ПРЕДУСМОТРЕННЫХ ЗАКОНОМ, IBM, РАЗРАБОТЧИКИ ПРОГРАММ И ПОСТАВЩИКИ НЕ ПРЕДОСТАВЛЯЮТ КАКИХ-ЛИБО ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ЭТИМ, ПОДРАЗУМЕВАЕМЫЕ ГАРАНТИИ СОБЛЮДЕНИЯ ПРАВ, КОММЕРЧЕСКОЙ ЦЕННОСТИ ИЛИ ПРИМЕНЕНИЯ ДЛЯ КАКИХ-ЛИБО КОНКРЕТНЫХ ЦЕЛЕЙ.

IBM, РАЗРАБОТЧИКИ ПРОГРАММ ИЛИ ПОСТАВЩИК НИ ПРИ КАКИХ УСЛОВИЯХ НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ЗА:

1. ПОТЕРЮ ИЛИ ПОВРЕЖДЕНИЕ ДАННЫХ;
2. ПРЯМОЙ, ЧАСТНЫЙ, СВЯЗАННЫЙ ИЛИ КОСВЕННЫЙ УЩЕРБ И ВЫЗВАННЫЙ ИМ ЭКОНОМИЧЕСКИЙ УЩЕРБ; ЛИБО
3. УПУЩЕННУЮ ВЫГОДУ, ПОТЕРЮ КЛИЕНТОВ, ДОХОДОВ, ДЕЛОВОЙ РЕПУТАЦИИ ИЛИ ИСТРАЧЕННЫЕ СБЕРЕЖЕНИЯ.

В НЕКОТОРЫХ ЮРИСДИКЦИЯХ НЕ ДОПУСКАЮТСЯ ИСКЛЮЧЕНИЯ ИЛИ ОГРАНИЧЕНИЯ ПРЯМОГО, СВЯЗАННОГО ИЛИ КОСВЕННОГО УЩЕРБА, ПОЭТОМУ НЕКОТОРЫЕ ИЛИ ВСЕ УКАЗАННЫЕ ВЫШЕ ОГРАНИЧЕНИЯ И ИСКЛЮЧЕНИЯ МОГУТ К ВАМ НЕ ОТНОСИТЬСЯ.

Условия и соглашения

Разрешение на использование этих публикаций предоставляется в соответствии с следующими условиями и соглашениями.

Личное использование: Вы можете воспроизводить эти публикации для личного, некоммерческого использования при условии сохранения информации об авторских правах. Данные публикации, а также любую их часть запрещается распространять, демонстрировать или использовать для создания других продуктов без явного согласия IBM.

Коммерческое использование: Вы можете воспроизводить, распространять и демонстрировать эти публикации в рамках своей организации при условии сохранения информации об авторских правах. Данные публикации, а также любую их часть запрещается воспроизводить, распространять, использовать для создания других продуктов и демонстрировать вне вашей организации, без явного согласия IBM.

На данные публикации, а также на содержащиеся в них сведения, данные, программное обеспечение и другую интеллектуальную собственность, не распространяются никакие другие разрешения, лицензии и права, как явные, так и подразумеваемые, кроме оговоренных в настоящем документе.

IBM сохраняет за собой право аннулировать предоставленные настоящим документом разрешения в том случае, если по мнению IBM использование этих публикаций может принести ущерб интересам IBM или если IBM будет установлено, что приведенные выше инструкции не соблюдаются.

Вы можете загружать, экспортировать и реэкспортировать эту информацию только в полном соответствии со всеми применимыми законами и правилами, включая все законы США в отношении экспорта.

IBM не несет ответственности за содержание этих публикаций. Публикации предоставляются на условиях ″как есть″, без предоставления каких-либо явных или подразумеваемых гарантий, включая, но не

ограничиваясь этим, подразумеваемые гарантии коммерческой ценности, отсутствия нарушений или применения для каких-либо конкретных целей.

Приложение. Примечания

Настоящая документация была разработана для продуктов и услуг, предлагаемых на территории США.

IBM может не предлагать продукты, услуги или функции, описанные в этом документе, в других странах. Для получения информации о продуктах, предлагаемых в вашей стране, обратитесь в местное представительство IBM. Ссылка на продукт, программу или услугу IBM не означает, что может быть использован только этот, продукт, программа или услуга. Может быть использован любой функционально эквивалентный продукт, программа или услуга, если его использование не ведет к нарушению прав IBM на интеллектуальную собственность. Однако в этом случае ответственность за проверку работы этих продуктов, программ и услуг возлагается на пользователя.

IBM может обладать патентами или находиться в процессе регистрации патентов, затрагивающих описанные в данном документе продукты и услуги. Предоставление вам настоящего документа не означает предоставления каких-либо лицензий на эти патенты. Запросы на приобретение лицензий можно отправлять по следующему адресу:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Запросы на получение лицензий, связанных с информацией, использующей двухбайтовые символы (DBCS), можно направлять в отдел интеллектуальной собственности IBM в вашей стране или в письменной форме по адресу:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

Следующий абзац не относится к Великобритании, а также к другим странам, в которых это заявление противоречит местному законодательству: ФИРМА INTERNATIONAL BUSINESS MACHINES CORPORATION ПРЕДОСТАВЛЯЕТ НАСТОЯЩУЮ ПУБЛИКАЦИЮ НА УСЛОВИЯХ “КАК ЕСТЬ”, БЕЗ КАКИХ-ЛИБО ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ЭТИМ, НЕЯВНЫЕ ГАРАНТИИ СОБЛЮДЕНИЯ ПРАВ, КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КАКОЙ-ЛИБО ЦЕЛИ. В некоторых странах запрещается отказ от каких-либо явных и подразумеваемых гарантий при заключении определенных договоров, поэтому данное заявление может не действовать в вашем случае.

В данной публикации могут встретиться технические неточности и типографские опечатки. В информацию периодически вносятся изменения, которые будут учтены во всех последующих изданиях настоящей публикации. IBM может вносить улучшения и/или изменения в продукт или продукты и/или программы, описанные в данной публикации, в любое время без дополнительного уведомления.

Все встречающиеся в данной документации ссылки на Web-сайты других компаний предоставлены исключительно для удобства пользователей и не являются рекламой этих Web-сайтов. Материалы, приведенные на этих Web-сайтах, не входят в состав материалов для данного продукта IBM, и ответственность за использование этих Web-сайтов несет пользователь.

IBM может использовать и распространять любую предоставленную вами информацию на свое усмотрение без каких-либо обязательств перед вами.

Для получения информации об этой программе для обеспечения: (i) обмена информацией между независимо созданными программами и другими программами (включая данную) и (ii) взаимного использования информации, полученной в ходе обмена, пользователи данной программы могут обращаться по адресу:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Такая информация может предоставляться на определенных условиях, включая, в некоторых случаях, уплату вознаграждения.

Описанная в этой информации лицензионная программа и все связанные с ней лицензионные материалы предоставляются IBM в соответствии с условиями Соглашения с заказчиком IBM, Международного соглашения о лицензии на программу IBM, Лицензионного соглашения о машинном коде IBM или любого другого эквивалентного соглашения.

Все приведенные показатели производительности были получены в управляемой среде. В связи с этим результаты, полученные в реальной среде, могут существенно отличаться от приведенных. Некоторые измерения могли быть выполнены в системах, находящихся на этапе разработки, поэтому результаты измерений, полученные в серийных системах, могут отличаться от приведенных. Более того, некоторые значения могли быть получены в результате экстраполяции. Реальные результаты могут отличаться от указанных. Пользователи, работающие с этим документом, должны удостовериться, что используемые ими данные применимы в имеющейся среде.

Информация о продуктах других изготовителей получена от поставщиков этих продуктов, из их официальных сообщений и других общедоступных источников. IBM не проводила тестирование этих продуктов и не может подтвердить точность данных о производительности и совместимости, а также других данных для продуктов других фирм. Запросы на получение дополнительной информации об этих продуктах должны направляться их поставщикам.

Все заявления, касающиеся намерений и планов IBM, могут изменяться и отзываться без предварительного уведомления, и отражают только текущие цели и задачи.

Все цены IBM указаны как розничные рекомендуемые IBM на данный момент. Они могут изменяться без предупреждения. Цены дилеров могут отличаться от указанных.

Эта информация предназначена только для целей планирования. Приведенная информация может измениться до того, как описанные в ней продукты станут доступными.

Данная документация содержит примеры данных и отчетов, применяемых в ежедневных деловых операциях. Для более наглядной демонстрации примеры включают имена отдельных людей, названия компаний и продуктов, а также торговые марки. Все имена и названия являются вымышленными; любые совпадения с именами, названиями и адресами реальных компаний случайны.

Лицензионное соглашение:

Данный документ содержит исходные тексты примеров программ, демонстрирующие технологии программирования на различных операционных платформах. Эти примеры можно копировать, изменять и распространять в любой форме без возмещения в адрес IBM, с целью разработки, применения, продвижения на рынке и распространения прикладных программ, поддерживающих интерфейсы прикладных программ операционных платформ, для которых они создаются. Примеры не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

Каждая полная или частичная копия, а также программа, включающая такую копию, должна содержать следующую информацию об авторских правах:

© (название вашей компании) (год). Часть кода данной программы предоставлена IBM Corp. в составе примеров программ. © Copyright IBM Corp. _год или годы_. All rights reserved.

При просмотре данного документа в электронном виде некоторые фотографии и цветные иллюстрации могут быть не показаны.

Информация об интерфейсе программирования

В данной материалах по IBM Toolbox for Java описаны программные интерфейсы, позволяющие заказчикам создавать программы с помощью IBM Toolbox for Java.

Товарные знаки

Ниже перечислены товарные знаки International Business Machines Corporation в Соединенных Штатах и/или других странах:

Advanced Function Presentation
Advanced Function Printing
AFP
AIX
alphaWorks
DB2
DB2 Universal Database
developerWorks
i5/OS
IBM
IPDS
iSeries
NetServer
PartnerWorld
SecureWay
System i
System i5
S/390
VisualAge
WebSphere

Adobe, эмблема Adobe, PostScript и эмблема PostScript являются зарегистрированными или обычными товарными знаками Adobe Systems Incorporated в США, других странах или и тех, и других.

Linux является зарегистрированным товарным знаком Linus Torvalds Group в США и/или других странах.

Microsoft, Windows, Windows NT и логотип Windows являются товарными знаками Microsoft Corporation в США и/или других странах.

Java и все товарные знаки с использованием Java являются товарными знаками Sun Microsystems, Inc. в США и/или других странах.

UNIX является зарегистрированным товарным знаком The Open Group в США и/или других странах.

Названия других фирм, продуктов или услуг могут быть товарными или сервисными знаками других фирм.

Условия и соглашения

Разрешение на использование этих публикаций предоставляется в соответствии с следующими условиями и соглашениями.

Личное использование: Вы можете воспроизводить эти публикации для личного, некоммерческого использования при условии сохранения информации об авторских правах. Данные публикации, а также любую их часть запрещается распространять, демонстрировать или использовать для создания других продуктов без явного согласия IBM.

Коммерческое использование: Вы можете воспроизводить, распространять и демонстрировать эти публикации в рамках своей организации при условии сохранения информации об авторских правах. Данные публикации, а также любую их часть запрещается воспроизводить, распространять, использовать для создания других продуктов и демонстрировать вне вашей организации, без явного согласия IBM.

На данные публикации, а также на содержащиеся в них сведения, данные, программное обеспечение и другую интеллектуальную собственность, не распространяются никакие другие разрешения, лицензии и права, как явные, так и подразумеваемые, кроме оговоренных в настоящем документе.

IBM сохраняет за собой право аннулировать предоставленные настоящим документом разрешения в том случае, если по мнению IBM использование этих публикаций может принести ущерб интересам IBM или если IBM будет установлено, что приведенные выше инструкции не соблюдаются.

Вы можете загружать, экспортировать и реэкспортировать эту информацию только в полном соответствии со всеми применимыми законами и правилами, включая все законы США в отношении экспорта.

IBM не несет ответственности за содержание этих публикаций. Публикации предоставляются на условиях ″как есть″, без предоставления каких-либо явных или подразумеваемых гарантий, включая, но не ограничиваясь этим, подразумеваемые гарантии коммерческой ценности, отсутствия нарушений или применения для каких-либо конкретных целей.

Напечатано в Дании