IBM Toolbox for Java

Содержание

IBM Toolbox for Java

IBM Toolbox for Java - это набор классов Java(TM), который позволяет создавать программы на Java для доступа к данным серверов iSeries. С помощью этих классов можно создавать приложения типа клиент-сервер, аплеты и сервлеты для работы с данными серверов iSeries. Можно также запускать приложения Java, в которых используются классы IBM Toolbox for Java, в виртуальной машине Java (JVM) систем iSeries.

В IBM Toolbox for Java в качестве точек доступа к системам применяются серверы хоста iSeries. Поскольку в IBM Toolbox for Java применяются встроенные средства Java для работы с соединениями, то для работы с IBM Toolbox for Java можно не применять IBM  iSeries Access for Windows. Каждый сервер связан с отдельным заданием сервера, которое обменивается данными по соединению с сокетом.

Для получения дополнительной информации о IBM Toolbox for Java воспользуйтесь панелью навигации или откройте следующие разделы:

Новое в версии V5R3
Информация о существенных изменениях, новых функциях и других заслуживающих внимания усовершенствованиях.

Как напечатать раздел IBM Toolbox for Java
Загрузите или просмотрите раздел IBM Toolbox for Java в формате PDF. Вы можете также загрузить категорию IBM Toolbox for Java в архивном пакете.

Поиск классов
Эта функция позволяет быстро и просто выполнять поиск классов по именам, просматривать классы в пакетах, а также работать с полным алфавитным списком классов IBM Toolbox for Java.

Установка и управление IBM Toolbox for Java
Этот раздел содержит информацию об управлении IBM Toolbox for Java в системе. Приведены инструкции по установке на рабочих станциях и серверах. Простые примеры программ помогут вам больше узнать об использовании классов IBM Toolbox for Java в приложениях.

Классы IBM Toolbox for Java
Этот раздел содержит информацию о различных классах в пакетах IBM Toolbox for Java, позволяющих работать с данными сервера iSeries. В разделе приведены описания, примеры программ и технические сведения, которые помогут вам при разработке программ с помощью IBM Toolbox for Java.

Создание панелей GUI с помощью Graphical Toolbox
Graphical Toolbox позволяет создавать пользовательские графические панели интерфейса Java, которые можно применять в приложениях и аплетах Java, а также встраиваемых модулях Навигатора iSeries.

JavaBean
Этот раздел содержит сведения о создании объектов JavaBean с помощью общих классов IBM Toolbox for Java, которые являются стандартными компонентами Javasoft JavaBean. Раздел содержит также примеры применения объектов JavaBean в программах.

JDBC
Этот раздел содержит сведения о поддержке JDBC в IBM Toolbox for Java. С помощью JDBC программы могут отправлять запросы на Языке структурных запросов (SQL) в базы данных серверов и обрабатывать полученные результаты.

Поддержка Proxy
Этот раздел содержит сведения о поддержке серверов Proxy в IBM Toolbox for Java, которая включает в себя шифрование данных с помощью протокола SSL.

Защита
Этот раздел посвящен защите соединений TCP/IP между клиентами и серверами с помощью Java Secure Socket Extension (JSSE) и IBM Toolbox for Java.

Системные свойства
Этот раздел содержит информацию о настройке различных параметров работы IBM Toolbox for Java, таких, как задание сервера Proxy или уровня трассировки, с помощью системных свойств. Системные свойства позволяют вносить изменения в конфигурацию во время выполнения, не перекомпилируя код.

IBM Toolbox for Java 2 Micro Edition
Этот новый компонент IBM Toolbox for Java позволяет создавать программы Java для различных беспроводных устройств. С помощью ToolboxME for iSeries беспроводные устройства могут напрямую обращаться к данным и ресурсам сервера iSeries.

Компоненты XML
Этот раздел содержит сведения о различных компонентах XML в IBM Toolbox for Java, упрощающих доступ и применение данных и функций iSeries в программах на Java.

Javadoc для IBM Toolbox for Java
Дополнительная информация о классах IBM Toolbox for Java приведена в справочной информации по Java.

Часто задаваемые вопросы (FAQ)
Этот раздел содержит ответы на вопросы об повышении производительности IBM Toolbox for Java, устранении неполадок, применении JDBC и многие другие вопросы.

Дополнительная информация включает следующие разделы:

– Раздел Советы по программированию содержит различную информацию о применении IBM Toolbox for Java
– Список примеров программ IBM Toolbox for Java
– Связанная информация, включая ссылки на дополнительную информацию о Java, сервлетах, XML и пр.

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

Новое в версии V5R3

Продукт IBM Toolbox for Java поставляется в нескольких вариантах:

- Лицензионная программа IBM Toolbox for Java, 5722-JC1, версия 5, выпуск 3 (V5R3), предназначенная для установки в OS/400 версии V5R1 и выше. Клиент IBM Toolbox for Java может подключаться к OS/400 версии V5R1 и выше.
- В состав операционной системы OS/400 входят классы IBM Toolbox for Java без поддержки графического интерфейса, оптимизированные для работы с виртуальной машиной Java (JVM) системы iSeries. Если вы не планируете применять возможности лицензионной программы по созданию графического интерфейса, вы можете воспользоваться этой версией IBM Toolbox for Java. За дополнительной информацией обратитесь к разделу Файлы Jar.
- IBM Toolbox for Java поставляется также с исходными файлами. Их и дополнительную информацию можно загрузить с Web-сайта JTOpen  .

Новые пакеты

Пакет com.ibm.as400.commtrace содержит набор классов, позволяющих программам на Java работать с данными трассировки соединений выбранных описаний линий локальных сетей (Ethernet и Token-Ring). Необработанные данные трассировки, сохраненные в файле вывода или двоичном файле, можно передавать, форматировать и анализировать (в отформатированном и необработанном виде), получая необходимую информацию. Можно также запустить любой класс в виде отдельной программы из командной строки.

Пакет com.ibm.as400.util содержит служебные классы, расширяющие возможности служебного пакета.

Новые классы

Стандартные пакеты IBM Toolbox for Java версии V5R3 содержат новые классы. Новые классы позволяют:

- С помощью класса HTMLDocument и существующих классов HTML IBM Toolbox for Java можно создавать страницы HTML и документы, содержащие теги форматирования объектов (FO) языка XSL. Теги XSL FO позволяют преобразовывать эти документы в документы других типов, такие как PDF.
- Класс IFSFileSystemView является графическим инструментом для работы с интегрированной файловой системой сервера. С его помощью можно создавать в программах на Java объекты javax.swing.JFileChooser, которые, как правило, применяются для поиска и выбора файлов. IFSFileSystemView заменил класс IFSFileDialog. IBM Toolbox for Java поддерживает класс IFSFileDialog, но, тем не менее, рекомендуется применять IFSFileSystemView.
- С помощью класса CommandHelpRetriever можно создавать документацию по командам CL в формате IBM. Класс CommandHelpRetriever можно запустить из командной строки или использовать в программе на Java.

Измененные классы

В IBM Toolbox for Java версии V5R3 изменены некоторые из старых классов. К числу изменений относится:

- Обновленные классы HTML упрощают создание исходных данных HTML и XSL FO и создание тегов заголовков страниц HTML.

Информация о расширенных функциях JDBC приведена в разделе Расширенная поддержка JDBC в IBM Toolbox for Java

Добавлен компонент XML

В IBM Toolbox for Java версии V5R3 появилась поддержка языка XPCML. Благодаря поддержке схем XML язык XPCML обладает более широкими функциональными возможностями и областью применения по сравнению с PCML. Например, с помощью XPCML можно задавать и передавать значения параметров программ, получать результаты работы программ iSeries в формате XPCML и выполнять другие операции.

Совместимость

В состав IBM Toolbox for Java больше не входит класс x4j400.jar (анализатор IBM XMLr). 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 
- Один из анализаторов XML, которые входят в состав OS/400 и расположены в каталоге /QIBM/ProdData/OS400/xml/lib

Продукт IBM Toolbox for Java теперь нельзя запустить в стандартной JVM продуктов Netscape(R) Navigator и Microsoft(R) Internet Explorer. Для того чтобы запустить в браузере аплет с классами Toolbox for Java, нужно установить специальный встраиваемый модуль, например Sun Java 2 Runtime Environment (JRE)  .

В состав IBM Toolbox for Java больше не входит файл data400.jar. Классы из этого файла включены в файл jt400.jar. Удалите файл data400.jar из переменной CLASSPATH.

Метод getObject() объектов ResultSet и CallableStatement теперь возвращает результат типа Integer в тех случаях, когда SQLType=SMALLINT. Ранее в таких случаях возвращался результат типа Short. Если вы пользуетесь методом readObject для чтения данных из столбцов типа SMALLINT, нужно модифицировать ваше приложение на Java с учетом изменения типа возвращаемого значения.

Теперь при усечении данных вместо ошибок выдаются предупреждения, не прерывающие работу программы.

Данный выпуск IBM Toolbox for Java не позволяет считывать некоторые объекты, сохраненные в виде двоичных файлов в версиях до V5R1.

Если вы пользуетесь протоколом Secure Sockets Layer (SSL) для шифрования данных, которыми обмениваются клиент с сервером, вам потребуется одно из следующего:

- Java Secure Socket Extension (JSSE)
- Объекты SSL, созданные с помощью лицензионной программы IBM iSeries Client Encryption (5722-CE2 или 5722-CE3) версии V5R1 или выше. Данная версия IBM Toolbox for Java не поддерживает объекты, созданные с помощью программы iSeries Client Encryption версий V4R5 и ниже.

Для применения всех классов IBM Toolbox for Java необходимо использовать платформу 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 JRE версии 1.4 или выше.

Дополнительная информация приведена в разделе Требования к OS/400 для запуска IBM Toolbox for Java.

Как узнать, что изменено

Все изменения в данном руководстве (за исключением изменений в документации по языку Java) отмечены следующим образом:

-  . Обозначает начало измененной информации.
- . Обозначает конец измененной информации.

Сведения о новой и измененной информации для данного выпуска приведены в разделе Информация для пользователей.

Как напечатать этот раздел

Для просмотра или загрузки документа в формате PDF щелкните на ссылке IBM Toolbox for Java в формате PDF (около 2,8 Мб).

Примечание: Раздел IBM Toolbox for Java содержит некоторые сведения, не вошедшие в документы PDF.

Сохранение файлов PDF

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

- Щелкните правой кнопкой мыши на файле PDF в браузере (щелкните правой кнопкой на приведенной выше ссылке).
- Выберите **Сохранить объект как...**, если вы используете браузер Internet Explorer. Выберите **Сохранить ссылку как...**, если вы используете браузер Netscape Communicator.
- Укажите каталог, в котором вы хотите сохранить документ.
- Нажмите **Сохранить**.

Загрузка программы Adobe Acrobat Reader

Для просмотра и печати этих документов PDF необходима программа Adobe Acrobat. Ее можно загрузить с Web-сайта фирмы Adobe (www.adobe.com/products/acrobat/readstep.html)  .

Загрузка информации об IBM Toolbox for Java в виде пакета

Раздел IBM Toolbox for Java можно загрузить в виде пакета, который будет содержать документацию по Java, с Web-сайта IBM Toolbox for Java and JTOpen  .

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

Установка и управление IBM Toolbox for Java

IBM Toolbox for Java упрощает создание аплетов, сервлетов и приложений Java, предназначенных для доступа к ресурсам, данным и программам серверов iSeries с рабочих станций.

Следующие сведения помогут вам установить продукт IBM Toolbox for Java и начать работу с ним:

Управление установкой
В этом разделе описаны различные способы установки и управления Toolbox for Java.

Установка IBM Toolbox for Java
В этом разделе перечислены требования к OS/400 и рабочей станции, которые должны быть выполнены для установки Toolbox for Java в среде клиент-сервер. Эти сведения помогут вам установить Toolbox for Java как на серверах iSeries, так и на рабочих станциях.

Свойства системы
В этом разделе приведены сведения о свойствах системы, связанных с настройкой IBM Toolbox for Java.

Примеры простых программ
В этом разделе описаны основы работы с IBM Toolbox for Java. С его помощью вы сможете создать собственную программу для Toolbox for Java на базе простых примеров. Примеры демонстрируют основные операции Toolbox for Java при работе с данными и службами сервера iSeries.

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. Клиентами могут быть персональные компьютеры, рабочие станции и системы iSeries. Не забывайте о том, что систему iSeries или ее раздел можно настроить в качестве клиента. В последнем случае IBM Toolbox for Java нужно установить в разделе сервера, относящемся к клиенту.

Предусмотрены различные способы установки 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 в клиентских системах, при которой для установки и управления IBM Toolbox for Java в каждой клиентской системе применяется класс AS400ToolboxInstaller
- 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 этого способа заключается в ускорении запуска приложений, использующих классы 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 в клиентских системах

С помощью класса AS400ToolboxInstaller можно выполнить массовую сетевую установку Toolbox for Java в клиентских системах. Поскольку на каждом клиенте при этом устанавливается отдельная копия Toolbox for Java, этот способ установки также позволяет быстро запускать приложения IBM Toolbox for Java в клиентских системах. Кроме того, данный метод позволяет автоматически обновлять все копии IBM Toolbox for Java на клиентах.

Основной недостаток этого метода заключается в необходимости создания и обслуживания процесса, использующего AS400ToolboxInstaller для установки Toolbox for Java на клиентах.

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

Однако этот метод установки обладает существенным недостатком: увеличивается время запуска приложений IBM Toolbox for Java на клиентах. Кроме того, переменная CLASSPATH клиентов в этом случае будет ссылаться на сервер. Обеспечить доступ к файлам сервера можно с помощью продукта iSeries NetServer, входящим в комплект поставки OS/400, или любым другим способом, позволяющим клиентам обращаться к серверу iSeries NetServer - например, с помощью продукта iSeries Access for Windows.

Установка IBM Toolbox for Java

Сначала следует выбрать метод установки. После этого нужно убедиться, что ваша среда отвечает следующим требованиям:

- Требования к OS/400
- Требования к рабочей станции

Установка IBM Toolbox for Java

После выбора способа установки и проверки требований можно начать установку IBM Toolbox for Java:

- Установка IBM Toolbox for Java на сервере
- Установка IBM Toolbox for Java на рабочей станции

Требования к OS/400 для работы с IBM Toolbox for Java

После выбора метода установки убедитесь, что ваша среда OS/400 отвечает следующим требованиям:

- Необходимые компоненты OS/400
- Зависимость от других лицензионных программ
- Совместимость с различными уровнями OS/400
- Внутренняя оптимизация при запуске на JVM OS/400
- Требования для запуска приложений ToolboxME for iSeries

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

Необходимые компоненты OS/400:

Для работы IBM Toolbox for Java в среде клиент/сервер необходимо задействовать пользовательский профайл QUSER, запустить серверы хоста и поддержку TCP/IP:

- Для запуска серверов хоста должен быть включен пользовательский профайл QUSER.
- Серверы хоста прослушивают сокеты и устанавливают соединения с клиентами. В базовую часть OS/400 входит компонент Серверы хоста (лицензионный продукт 5722SS1). Дополнительные сведения приведены в разделе Управление серверами хоста.
- В OS/400 входит поддержка TCP/IP, позволяющая подключать сервер к сети. Дополнительные сведения приведены в разделе TCP/IP.

Запуск необходимых компонентов OS/400

Для запуска необходимых компонентов OS/400 выполните следующие действия в командной строке iSeries:

1. Убедитесь в том, что включен пользовательский профайл QUSER.
2. Запустите серверы хоста OS/400 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**.

Проверка наличия IBM Toolbox for Java на сервере:

Многие серверы iSeries поставляются с предустановленным лицензионным продуктом IBM Toolbox for Java.

Для того чтобы определить, установлен ли продукт IBM Toolbox for Java, выполните следующие действия:

1. Запустите Навигатор iSeries и войдите в систему.
2. В **дереве функций** (левая панель) откройте систему, затем **Настройка и обслуживание**.
3. Откройте **Программное обеспечение**, затем **Установленные продукты**.
4. Посмотрите, есть ли в столбце **Продукт** панели **Сведения** значение 5722jc1. 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, его можно установить.

Если установлена предыдущая версия IBM Toolbox for Java, ее необходимо удалить, а затем установить лицензионный продукт IBM Toolbox for Java. Для того чтобы предотвратить возможные неполадки, перед удалением текущей версии IBM Toolbox for Java рекомендуется создать ее резервную копию.

Проверка состояния профайла QUSER:

Серверы хоста OS/400 запускаются с профайлом пользователя QUSER, поэтому в первую очередь необходимо убедиться, что профайл QUSER включен.

Проверка состояния профайла QUSER

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

1. В командной строке iSeries введите `DSPUSRPRF USRPRF(QUSER)` и нажмите **Enter**.

2. Убедитесь, что в поле **Состояние** указано значение *ENABLED. Если в поле указано другое состояние, измените профайл QUSER.

Изменение пользовательского профайла QUSER:

Если профайл QUSER не находится в состоянии *ENABLED, вы должны разблокировать его для запуска OS/400 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**.

Теперь пользовательский профайл QUSER готов к запуску OS/400 Host Servers.

Зависимость от других лицензионных программ:

Для применения некоторых функций IBM Toolbox for Java требуется установить дополнительные лицензионные программы. Сведения об этом приведены ниже.

Программа просмотра буферных файлов

Если вы собираетесь использовать функцию просмотра буферных файлов IBM Toolbox для 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:

- Лицензионная программа IBM HTTP Server for iSeries, 5722-DG1
- Компонент 34 OS/400 (Digital Certificate Manager)
- IBM Cryptographic Access Provider 128-bit for iSeries, 5722-AC3
- iSeries Client Encryption (128-bit), 5722-CE3

Для работы с IBM Toolbox for Java версии V5R3 требуется установить продукт iSeries Client Encryption версии V5R1, V5R2 или V5R3.

Дополнительная информация об SSL приведена в разделе “Сравнение Secure Sockets Layer и Java Secure Socket Extension” на стр. 349.

Для работы с аплетами, сервлетами, SSL и продуктом AS400ToolboxInstaller требуется установить сервер HTTP.

Если вы планируете применять в системе iSeries аплеты, сервлеты, SSL или класс AS400ToolboxInstaller, необходимо настроить в этой системе сервер HTTP и установить файлы классов. Дополнительные сведения о сервере IBM HTTP Server приведены в руководстве IBM HTTP Server for AS/400 Webmaster’s Guide, GC41-5434, опубликованном по следующему адресу:

http://www.ibm.com/eserver/iseries/products/http/docs/doc.htm . 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.

Совместимость с разными версиями OS/400:

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.

Запуск IBM Toolbox for Java V5R3 на серверах

Для установки IBM Toolbox for Java (лицензионная программа 5722-JC1 V5R3M0) в системе iSeries должна быть установлена одна из следующих операционных систем:

- OS/400 V5R3
- OS/400 V5R2
- OS/400 V5R1

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.

На клиенте и сервере, между которыми устанавливается соединение, могут быть установлены различные версии IBM Toolbox for Java. Для того чтобы с помощью IBM Toolbox for Java версии V5R3 можно было обращаться к данным и ресурсам системы iSeries, **на целевом сервере** должна быть установлена любой из следующих операционных систем:

- OS/400 V5R3
- OS/400 V5R2
- OS/400 V5R1

В следующей таблице приведены требования по совместимости для установки IBM Toolbox for Java и подключения к разным версиям OS/400.

Примечание: IBM Toolbox for Java не поддерживает совместимость со следующими версиями. IBM Toolbox for Java нельзя установить на сервере или подключить к серверу, на котором установлена более поздняя версия OS/400. Например, IBM Toolbox for Java, входящий в комплект поставки OS/400 V5R1, нельзя установить или подключить к системе, работающей под управлением OS/400 V5R3.

Программы	Поставляется с OS/400	Устанавливается в OS/400	Подключается к OS/400
5722-JC1 V5R1M0	V5R1	V4R4 и выше	V4R3 и выше
5722-JC1 V5R2M0	V5R2	V4R5 и выше	V4R5 и выше
5722-JC1 V5R3M0	V5R3	V5R1 и выше	V5R1 и выше

Оптимизация при работе с JVM системы iSeries:

Для того чтобы работа классов IBM Toolbox for Java отвечала требованиям пользователей OS/400, предусмотрен набор функций внутренней оптимизации. Эта оптимизация влияет на работу IBM Toolbox for Java только в случае применения JVM iSeries.

Необходимо помнить, что программы на Java применяют внутреннюю оптимизацию только при совпадении версии IBM Toolbox for Java с версией OS/400, установленной на сервере. Внутренняя оптимизация:

- Вход в систему: Если в объекте AS400 не указан ИД пользователя или пароль, применяется ИД пользователя и пароль текущего задания
- Вызов API OS/400 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: Для повышения производительности задайте в свойстве драйвера JDBC применение исходного драйвера для случаев, когда программа на Java и файл базы данных расположены в одной системе iSeries.

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

Требования к обеспечению оптимизации

В следующей таблице показано, какие версии IBM Toolbox for Java и операционной системы OS/400 поддерживают внутреннюю оптимизацию. Таблица содержит информацию о совместимости, касающуюся только применения внутренней оптимизации. Общие сведения о совместимости приведены в разделе Совместимость с разными версиями OS/400.

Версия OS/400	**IBM Toolbox for Java LPP, необходимый для применения внутренней оптимизации**		
V5R1	5722-JC1 V5R1M0		
V5R2		5722-JC1 V5R2M0	
V5R3			5722-JC1 V5R3M0

Для повышения производительности следует использовать файл jar, содержащий функции внутренней оптимизации OS/400. Дополнительные сведения приведены в примечании 1 к разделу Файлы Jar.

При несовпадении версий IBM Toolbox for Java и OS/400 внутренняя оптимизация недоступна. В этом случае IBM Toolbox for Java работает так же, как на клиенте.

Требования ToolboxME for iSeries:

Для того чтобы вы могли разрабатывать и запускать приложения ToolboxME for iSeries, рабочая станция, беспроводное устройство и сервер должны удовлетворять определенным требованиям (они перечислены ниже). Несмотря на то, что компонент IBM Toolbox for Java 2 Micro Edition считается частью IBM Toolbox for Java, он не входит в состав лицензионного продукта.

ToolboxME for iSeries (jt400Micro.jar) входит в состав версии Toolbox for Java с открытым исходным текстом, которая называется JTOpen. Вы должны отдельно загрузить и настроить ToolboxME for iSeries, содержащийся в JTOpen.

Требования

Для работы с ToolboxME for iSeries необходимо, чтобы рабочая станция, беспроводное устройство Tier0 и сервер удовлетворяли следующим требованиям.

Требования к рабочей станции

Требования к рабочей станции для разработки приложений ToolboxME for iSeries:

- Java 2 Platform, Standard Edition, версия 1.3 или выше
- Виртуальная машина Java для беспроводных устройств
- Симулятор или эмулятор беспроводного устройства

Требования к беспроводному устройству

Единственное условие для запуска приложений ToolboxME for iSeries на устройстве Tier0 - применение виртуальной машины Java для беспроводных устройств.

Требования к серверу

Ниже перечислены требования к серверу для работы с приложениями ToolboxME for iSeries:

- Класс MEServer, который входит в IBM Toolbox for Java или в последнюю версию JTOpen
- Требования к OS/400 для работы с IBM Toolbox for Java

Требования к рабочей станции для установки IBM Toolbox for Java

После выбора метода установки, нужно убедиться, что рабочая станция отвечает следующим требованиям:

- Требования для запуска приложений Java
- Требования для запуска аплетов Java
- Требования для разработки приложений ToolboxME for iSeries
- Требования Swing

Примечание: Перед установкой IBM Toolbox for Java убедитесь, что выполнены соответствующие требования к OS/400.

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 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(R) 2000
  - Windows(R) XP
  - AIX версии 4.3.3.1
  - Sun Solaris(TM) версии 5.7
  - OS/400 версии 5, выпуска 1 или выше
  - 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 версии 4.7 со встраиваемым модулем Java версии 1.3 или выше

  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 Toolbox for Java, нужно установить специальный встраиваемый модуль, например Sun Java 2 Runtime Environment (JRE) .
- Установленный и настроенный стек TCP/IP.

- Соединение с сервером с OS/400 версии V5R1 или выше

Требования к рабочей станции для применения компонента Swing продукта 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. 

Установка IBM Toolbox for Java на сервере iSeries

Продукт IBM Toolbox for Java нужно устанавливать на сервере iSeries только в тех случаях, когда сервер или хотя бы один из его разделов выполняют функции клиента.

Примечание: Оригинальная версия IBM Toolbox for Java входит в комплект поставки OS/400. По этой причине, если IBM Toolbox for Java применяется только в системе iSeries, лицензионный продукт можно не устанавливать. Дополнительные сведения о версии Toolbox for Java, поставляемой вместе с OS/400, приведены в разделе Файлы Jar: примечание 1.

Перед установкой IBM Toolbox for Java нужно убедиться, что применяемая версия OS/400 отвечает требованиям для применения Toolbox for Java. Некоторые серверы поставляются с заранее установленным продуктом IBM Toolbox for Java. Рекомендуется выяснить, установлен ли в системе лицензионный продукт IBM Toolbox for Java .

Установка IBM Toolbox for Java

Установить IBM Toolbox for Java можно с помощью Навигатора iSeries или из командной строки.

Установка IBM Toolbox for Java с помощью Навигатора iSeries

Для установки Toolbox for Java с помощью Навигатора iSeries выполните следующие действия:

1. Запустите Навигатор iSeries и войдите в систему.
2. Откройте **Мои соединения** в левой панели (Дерево функций).
3. В разделе **Мои соединения** щелкните правой кнопкой мыши на системе, в которой нужно установить Toolbox for Java.
4. Выберите пункт **Выполнить команду**.
5. В окне **Восстановить лицензионную программу (RSTLICPGM)** введите следующие сведения, а затем нажмите кнопку **OK**:
   - Продукт: 5722JC1
   - Устройство: имя нужного устройства или файла сохранения

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

С помощью Навигатора iSeries можно просмотреть результаты выполнения соответствующей команды централизованного управления:

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.

Установка IBM Toolbox for Java с помощью командной строки

Для установки Toolbox for Java из командной строки iSeries выполните следующие действия:

1. Выполните команду Перейти к меню. Введите **GO MENU(LICPGM)** и нажмите **ENTER**.
2. Выберите опцию **11. Установить лицензионную программу**.
3. Выберите **5722-JC1 IBM Toolbox for Java**.

Дополнительные сведения об установке лицензионных программ приведены в разделе Управление программным обеспечением и лицензионными программами.

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 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. В этом случае на сервере должен быть установлен продукт iSeries Netserver.

В этом разделе приведены инструкции по копированию файлов с классами на рабочую станцию. Инструкции по применению переменной CLASSPATH приведены в документации к операционной системе, установленной на рабочей станции, а также на сайте Sun Java  в Internet.

Примечание: Для применения классов IBM Toolbox for Java при работе с приложением система должна соответствовать требованиям к OS/400.

Файлы классов 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:
   - Подключить сетевой диск с помощью iSeries Access и скопировать файл.
   - 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 Toolbox for Java с открытым исходным текстом, которая называется JTOpen. Сведения о JTOpen приведены на сайте IBM Toolbox for Java и JTOpen  .

Файлы Jar:

IBM Toolbox for Java поставляется в виде набора файлов jar. 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.

Пакет или функция IBM Toolbox for Java	**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. 250	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
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

Пакет или функция IBM Toolbox for Java	Файлы Jar, которые нужно добавить в переменную CLASSPATH
Системный отладчик iSeries [Примечание 3]	jt400.jar (клиентский) [Примечание 1] и tes.jar
ToolboxME for iSeries	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]

Примечание 1: Некоторые классы IBM Toolbox for Java расположены в нескольких файлах jar:

- **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 поставляется для совместимости с предыдущими выпусками IBM Toolbox for Java.
- **jt400Access.zip** - Те же классы, что и в jt400.jar, за исключением классов графического интерфейса. jtAccess400.zip поставляется для совместимости с предыдущими выпусками 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. Классы внутренней оптимизации - это набор из менее чем 20 классов, которые повышают эффективность работы программ на Java в JVM системы iSeries за счет средств iSeries. Класс jt400Native.jar рекомендуется применять вместо класса jt400.jar при работе с JVM системы iSeries. Файл jt400Native.jar поставляется вместе с операционной системой OS/400. Он расположен в каталоге /QIBM/ProdData/OS400/jt400/lib.
- **jt400Native11x.jar** - Только классы внутренней оптимизации. Если при работе с JVM системы iSeries вы хотите использовать файл jt400.jar, укажите файл jt400Native11x.jar в переменной CLASSPATH вместо файла jt400Native.jar. Файл jt400Native11x.jar поставляется вместе с операционной системой OS/400. Он расположен в каталоге /QIBM/ProdData/OS400/jt400/lib.

Примечание 2: При применении анализатора XML и обработчика XSLT убедитесь, что они совместимы с JAXP. Дополнительная информация приведена на следующей странице:

“Анализатор XML и обработчик XSLT” на стр. 412

Примечание 3: Для применения CommandPrompter, PDML и системного отладчика iSeries System необходим следующий файл jar, который не входит в IBM Toolbox for Java: jhall.jar. Дополнительные сведения о загрузке файла jhall.jar приведены на Web-сайте Sun JavaHelp(TM)  .

Примечание 4: файл util400.jar содержит классы для iSeries, позволяющие форматировать ввод и работать с приглашением командной строки. Этот файл необходим для применения класса CommandPrompter. Для применения PDML файл util400.jar не требуется, однако он содержит некоторые полезные функции.

Примечание 5: файл jui400.jar содержит классы, необходимые для работы с GUI источника данных JDBC. Файл jt400.jar (Примечание 1) содержит классы, необходимые для применения всех остальных функций JDBC.

Примечание 6: Файл jt400Mri_xx_yy.jar содержит преобразованные сообщения, в том числе текст сообщений об исключительных ситуациях и окон диалога, а также вывод других стандартных сообщений. В файле jt400Mri_lang_cntry.jar, lang = Код языка ISO, а cntry = Код страны или региона ISO, применяемый для перевода содержимого. В некоторых случаях Код страны или региона ISO не применяется. При установке версии IBM Toolbox for Java с поддержкой определенного национального языка автоматически

устанавливается соответствующий файл jt400Mri_lang_cntry.jar. Для неподдерживаемых языков по умолчанию устанавливается английская версия продукта, содержащаяся в файлах jar IBM Toolbox for Java.

- Например, при установке немецкой версии лицензионной программы 5722-JC1 автоматически устанавливается файл jar для немецкого языка, 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.

Примечание 7: Преобразование файла PCML в двоичный формат во время разработки обеспечивает следующие преимущества:

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. За дополнительной информацией обратитесь к разделу Создание вызовов программ iSeries с помощью PCML.

Примечание 8: Вместо файла data400.jar воспользуйтесь файлами jt400.jar и jt400Native.jar. Файл data400.jar содержит классы выполнения PCML, которые также входят в jt400.jar и jt400Native.jar (Примечание 1). Файл data400.jar поставляется для совместимости с предыдущими выпусками IBM Toolbox for Java.

Примечание 9: Копии классов ReportWriter содержатся в следующих файлах jar:

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

Если приложение сохраняет поток данных PCL в буферном файле iSeries, необходимо обеспечить возможность работы с классами доступа, воспользовавшись соответствующим файлом jar (Примечание 1). Для записи данных PCL в буферный файл необходимы классы AS400, OutputQueue, PrintParameterList и SpooledFileOutputStream. Дополнительная информация приведена в разделе Классы ReportWriter.

Примечание 10: jt400Micro.jar не содержит классы, необходимые для работы MEServer. Эти классы содержатся в файлах jt400.jar и jt400Native.jar (Примечание 1). Файл jt400Micro.jar можно загрузить только с Web-сайта IBM Toolbox for Java and JTOpen .

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
- Свойства CommandCall/ProgramCall
- Свойства FTP
- Свойства соединения

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.SecureAS400.proxyEncryptionMode	Указывает, какие данные, передаваемые через Proxy, шифруются с помощью SSL. Допустимые значения: • 1 = Данные, передаваемые между клиентом и сервером Proxy • 2 = Данные, передаваемые между сервером Proxy и системой iSeries • 3 = Данные, передаваемые между клиентом Proxy и системой iSeries через сервер Proxy
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. Полный список категорий трассировки определен в классе Trace
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

```
#=========================================================#
# 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 указывает, какие данные, передаваемые
# через Proxy, шифруются с помощью SSL. Допустимые значения:
#   1 - Данные, передаваемые между клиентом и сервером Proxy
#   2 - Данные, передаваемые между сервером Proxy и системой AS/400
#   3 - Данные, передаваемые между клиентом Proxy и AS/400 через сервер Proxy
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=1

# Данное системное свойство задает периодичность выполнения
# процедуры 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.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, какие данные, передаваемые
      // через Proxy, шифруются с помощью SSL.  Допустимые значения:
      //   1 - Данные, передаваемые между клиентом и сервером Proxy
      //   2 - Данные, передаваемые между сервером Proxy и сервером iSeries
      //   3 - Оба типа данных, перечисленных выше
      put("com.ibm.as400.access.SecureAS400.proxyEncryptionMode", "1");

      // Данное системное свойство указывает, с какой периодичностью
      // сервер Proxy выполняет поиск простаивающих соединений (в с.).
      // Сервер Proxy запускает отдельную нить для поиска клиентов,
      // 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");
   }
}
```

Классы IBM Toolbox for Java

Классы IBM Toolbox for Java объединяются (как и любые классы Java) в пакеты. 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.

Следующий список ссылок позволяет найти информацию о классах различных пакетов Toolbox for Java:

- Классы доступа предназначены для доступа к ресурсам системы iSeries и управления ими
- Раздел “Классы Commtrace” на стр. 177 содержит информацию о работе с данными трассировки линий Ethernet и Token-Ring
- Классы HTML предназначены для создания форм и таблиц HTML
- Классы Micro позволяют создавать программы на Java, обеспечивающие прямой доступ беспроводных устройств к службам и данным сервера iSeries
- Классы ReportWriter позволяют создавать отформатированные документы на основе источников данных XML
- Классы ресурсов предоставляют универсальные средства для доступа к ресурсам сервера iSeries и работы с ними
- Классы защиты позволяют создавать защищенные соединения с сервером и идентифицировать пользователей, работающих на сервере iSeries
- Классы сервлетов предназначены для получения и форматирования данных в сервлетах Java
- Классы утилит предназначены для выполнения административных задач. К ним относятся классы AS400ToolboxInstaller и AS400JarMaker
- Классы графического интерфейса предназначены для визуального представления данных и работы с графическими данными

Классы доступа

Классы доступа IBM Toolbox for Java обеспечивают доступ к данным и ресурсам системы iSeries. Эти классы работают с серверами iSeries и предоставляют интерфейс Internet для доступа и обновления данных и ресурсов сервера.

Доступ к ресурсам сервера обеспечивается следующими классами:

- AS400 - управляет информацией входа в систему, устанавливает и поддерживает соединения через API сокетов и отвечает за обмен данными
- SecureAS400 - поддерживает шифрование данных, передаваемых между сервером и объектом AS400
- AS400JPing - позволяет программе на Java проверять работу сервера и его служб, а также получать информацию о портах этих служб
- BidiTransform - преобразует двунаправленный текст между различными форматами

- Классы кластерных хэш-таблиц - позволяют программам на Java совместно использовать и копировать изменяющиеся данные между узлами в кластерных хэш-таблицах высокой готовности
- Вызов команды - запускает команды iSeries в пакетном режиме
- Пул соединений - управляет пулом объектов AS400,применяемым для совместного использования соединений с сервером iSeries
- Область данных - обеспечивает создание, использование и удаление областей данных
- Преобразование и описание данных - преобразует данные, обеспечивает работу с ними и позволяет описывать формат записи для буфера данных
- Очереди данных - обеспечивает создание, использование, изменение и удаление очередей данных
- Цифровые сертификаты - управляет цифровыми сертификатами в iSeries
- Переменная среды - управляет переменными среды iSeries
- Протокол событий - обеспечивает регистрацию исключительных ситуаций и сообщений, независимую от устройства вывода
- Исключительные ситуации - обеспечивает выдачу стандартных сигналов об ошибках при сбоях устройств, ошибках программ и прочих неполадках
- FTP - предоставляет интерфейс для работы с функциями FTP
- Интегрированная файловая система - управляет доступом к файлам, открытием файлов, открытием потоков ввода-вывода, а также чтением содержимого каталогов
- Вызов приложений Java - запускает программу на Java в виртуальной машине Java для iSeries
- JDBC - предоставляет доступ к данным iSeries через DB2 UDB
- Задания - обеспечивает доступ к заданиям и протоколам заданий системы iSeries
- Сообщения - обеспечивают доступ к сообщениям и очередям сообщений на сервере iSeries
- Конфигурация NetServer - позволяет просматривать и изменять состояние и конфигурацию iSeries NetServer
- Права доступа - позволяет просматривать и изменять права доступа к объектам сервера iSeries
- Печать - управляет ресурсами печати AS/400
- Лицензия на продукт - управляет лицензиями на программные продукты iSeries
- Вызов программ - позволяет вызывать программы iSeries
- Путь к объектам QSYS - обеспечивает доступ к объектам интегрированной файловой системы iSeries
- Доступ на уровне записей - позволяет создавать, считывать, обновлять и удалять файлы и элементы файлов в системе iSeries
- Вызов служебных программ - позволяет вызывать служебные программы iSeries
- Состояние системы - позволяет просматривать состояние системы и обеспечивает доступ к данным в системном пуле
- Системные значения - позволяет считывать и изменять системные значения и сетевые атрибуты
- Трассировка (обслуживание) - регистрирует точки трассировки и диагностические сообщения
- Пользователи и группы - предоставляет доступ к профайлам пользователей и групп iSeries
- Пользовательское пространство - обеспечивает доступ к пользовательскому пространству iSeries

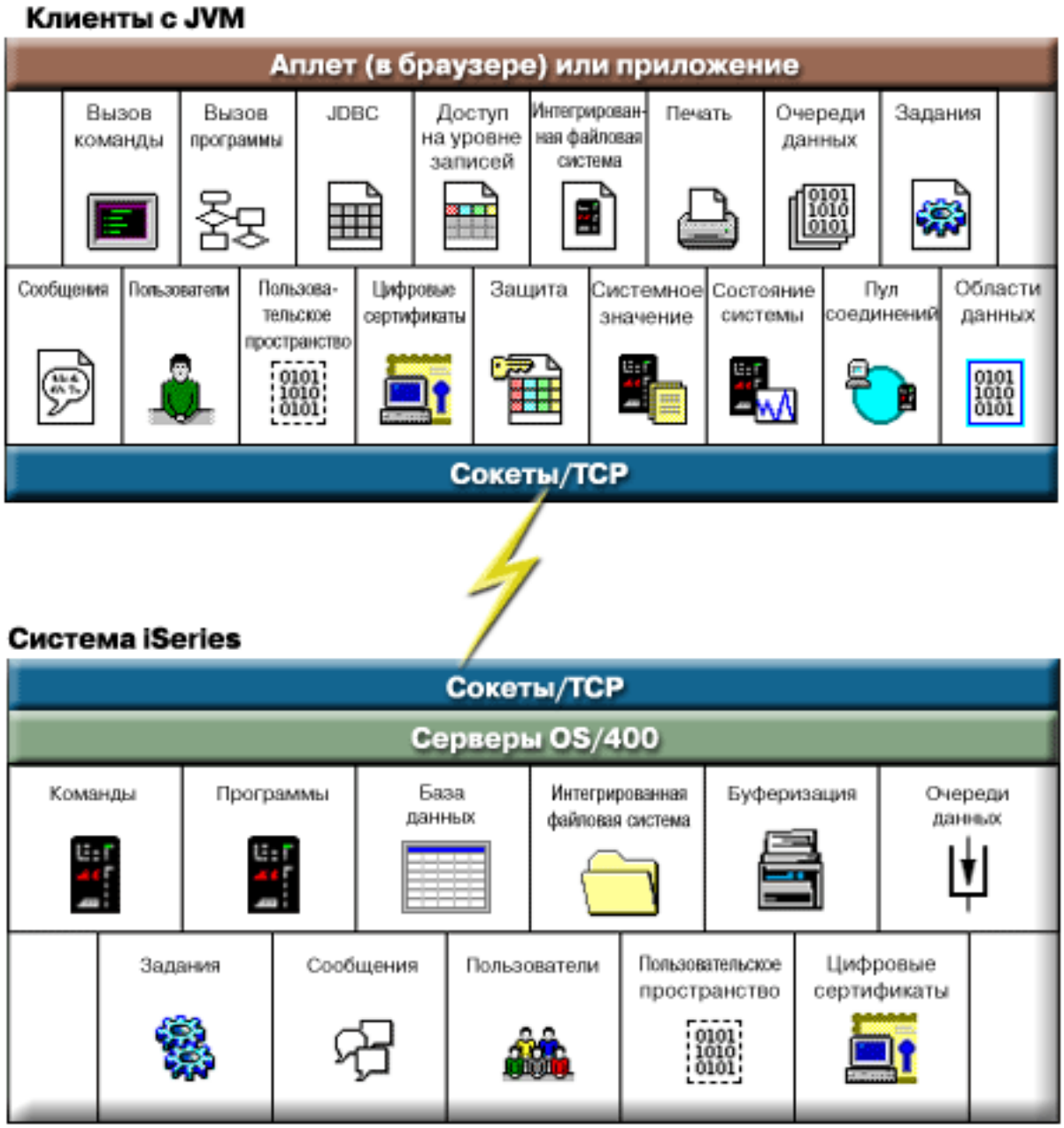
Примечание: IBM Toolbox for Java содержит второй тип классов, который называется классы ресурсов, для работы с объектами и списками iSeries. Набор классов ресурсов предоставляет общую среду и согласованный интерфейс для работы с различными объектами и списками iSeries. Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий приложению.

Точки доступа сервера

Классы доступа IBM Toolbox for Java предоставляют функциональные возможности, аналогичные API IBM iSeries Access для Windows. Однако для работы с классами доступа устанавливать iSeries Access для Windows необязательно.

Классы доступа применяют существующие серверы iSeries в качестве точек доступа. Каждый сервер работает в отдельном задании системы iSeries и обеспечивает обмен потоками данных с помощью API сокетов.

Рис. 1: Точки доступа сервера



Подробное описание рисунка 1: Точки доступа сервера (rzahh501.gif): IBM Toolbox for Java: Точки доступа сервера

На этом рисунке продемонстрировано применение соединений с сокетами в классах доступа продукта IBM Toolbox for Java для работы с данными и службами сервера iSeries.

Описание

Рисунок состоит из следующих компонентов:

- Прямоугольник в верхней части рисунка представляет одного или нескольких клиентов, на каждом из которых запущена Виртуальная машина Java. В описании указано, что на клиенте запущен аплет в браузере или приложение на Java, и клиент подключен к серверу iSeries по соединению с сокетом или соединению TCP.
- Нижний прямоугольник представляет систему iSeries. В описании указано, что он представляет один или несколько серверов iSeries с операционной системой OS/400, которые подключены к клиенту по соединению с сокетом или соединению TCP.
- Молния, соединяющая два прямоугольника, представляет активное соединение с сокетом или соединение TCP, служащее для передачи информации между клиентом и серверами iSeries.

В пакете классов доступа IBM Toolbox for Java клиента предусмотрены различные функции для работы с данными с службами сервера iSeries, в том числе:

- Вызов команд
- Вызов программ
- JDBC
- Доступ на уровне записей
- Интегрированная файловая система
- Печать
- Очереди данных
- Задания
- Сообщения
- Пользователи
- Пользовательское пространство
- Цифровые сертификаты
- Защита
- Системное значение
- Состояние системы
- Пул соединений
- Области данных

В системе iSeries предусмотрены различные типы данных и служб, для работы с которыми могут применяться классы доступа Toolbox for Java:

- Команды
- Программы
- База данных
- Интегрированная файловая система
- Буфер
- Очереди данных
- Задания
- Сообщения
- Пользователи
- Пользовательское пространство
- Цифровые сертификаты

Класс AS400

Класс AS400 управляет следующими функциями:

- Набором соединений с заданиями сервера iSeries через сокеты.
- Входом в систему сервера. К функциям объекта относится выдача приглашений для идентификации пользователя, кэширование паролей и поддержка идентификаторов пользователей по умолчанию.

Объект AS400 необходим программе на Java для работы с классами, обращающимися к системе iSeries. Например, он применяется объектом CommandCall для передачи команд на сервер iSeries.

При работе под управлением виртуальной машины Java для iSeries работа объекта AS400 с соединениями, идентификаторами пользователей и паролями изменяется. Более подробная информация приведена в разделе Виртуальная машина Java для iSeries.

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). Дополнительная информация о JGSS приведена в разделе Документация по защите J2SDK версии 1.4  .

Управление соединениями с iSeries с помощью объекта AS400 описано в разделе Управление соединениями. Информация о том, как сократить время первого соединения с помощью пула соединений, приведена в разделе Пул соединений AS400.

Класс AS400 содержит следующие функции входа в систему:

- Идентификация пользователя
- Получение кратковременного одноразового разрешения и идентификация связанного пользовательского профайла
- Установка кратковременного одноразового разрешения
- Управление идентификаторами пользователей по умолчанию
- Кэширование паролей
- Выдача приглашения для ввода идентификатора пользователя
- Изменение пароля
- Получение версии и выпуска системы iSeries

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

Управление идентификаторами пользователей по умолчанию:

Для сокращения числа операций входа в систему можно применять ИД пользователя по умолчанию. Программа 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 имя 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(). Если не задан идентификатор по умолчанию и имя пользователя не указано приложением Java, объект AS400 выдает приглашение для ввода идентификатора при каждом подключении к серверу iSeries.

Все объекты AS400, представляющие одну систему iSeries в виртуальной машине Java, применяют один ИД пользователя по умолчанию.

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
                    // для одной системы iSeries.
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);
```

Информация об ИД пользователя по умолчанию удаляется при удалении последнего объекта AS400 во время сбора мусора.

Применение кэша паролей: Кэширование паролей позволяет IBM Toolbox for Java сохранять информацию об идентификаторах пользователей и паролях, чтобы пользователю не нужно было вводить пароль при каждом подключении. Методы объекта AS400 позволяют выполнить следующие действия:

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

Кэш паролей относится ко всем объектам AS400, представляющим сервер iSeries в виртуальной машине Java. Java не обеспечивает общего использования данных несколькими виртуальными машинами, поэтому пароли, находящиеся в кэше одной виртуальной машины, недоступны остальным виртуальным машинам. Кэш очищается во время сбора мусора при удалении последнего объекта AS400. Пользователь может включить кэширование пароля с помощью переключателя в окне диалога Вход в систему. Программы Java могут передавать ИД и пароль пользователя во время создания объекта AS400. Пароли, передаваемые 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: Вывод приглашения для ввода имени пользователя и пароля:

- Происходит при подключении к системе iSeries
- 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 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.

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**				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.
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.

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**		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.
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

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 их можно зашифровать с помощью Secure Sockets Layer (SSL). Объект SecureAS400 позволяет указать, какие данные должны передаваться в зашифрованном виде. 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.

Класс SecureAS400 - это подкласс класса AS400.

Для настройки защищенного соединения с сервером создайте экземпляр объекта SecureAS400 одним из следующих способов:

- SecureAS400(String systemName, String userID) запрашивает идентификационную информацию при входе в систему
- SecureAS400(String systemName, String userID, String password) не запрашивает информацию у пользователя при входе в систему

Ниже приведен пример запуска команд в системе iSeries с помощью класса CommandCall и защищенного соединения:

```
 // 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();
```

AS400JPing

С помощью класса AS400JPing программа на Java может отправлять запрос на серверы хоста и составлять список активных служб и занятых портов. Такой запрос можно отправить из командной строки с помощью класса JPing.

Класс AS400JPing содержит следующие методы:

- Проверка связи с сервером
- Проверка работы конкретной службы сервера
- Установка объекта PrintWriter, на котором следует вести журнал проверки связи
- Установка тайм-аута проверки связи

Пример: Применение класса AS400JPing

В приведенном ниже примере показано, как с помощью класса AS400JPing программа на Java может проверить связь со службой удаленных команд iSeries:

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

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

Класс BidiTransform

Класс AS400BidiTransform поддерживает преобразования формата, позволяющие преобразовывать двунаправленный текст в формате iSeries (после преобразования в Unicode) в двунаправленный текст в формате Java, а также обратно.

Класс AS400BidiTransform позволяет:

- Получать и устанавливать системный CCSID
- Получать и устанавливать тип строки данных iSeries
- Получать и устанавливать тип строки данных Java
- Преобразовывать данные из формата Java в формат iSeries
- Преобразовывать данные из формата iSeries в формат Java

Пример: Применение класса AS400BidiTransform

Следующий пример иллюстрирует преобразование двунаправленного текста с помощью класса AS400BidiTransform:

```
// Преобразование данных из формата Java в формат iSeries:
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.

Классы ClusteredHashTable

Классы кластерных хэш-таблиц (ClusteredHashTable) позволяют программам на Java работать с данными в режиме совместного использования и копировать эти данные в непостоянную память на узлах кластера с помощью кластерных хэш-таблиц. Перед началом работы с классами ClusteredHashTable убедитесь, что вы можете сохранять данные в непостоянной памяти. Копируемые данные не шифруются.

Примечание: В данной публикации предполагается, что вы знакомы со стандартными принципами работы и терминологией кластеров iSeries. Дополнительная информация о кластерах и работе с ними приведена в разделе Кластеры.

Для работы с классом ClusteredHashTable необходимо определить и активизировать кластер систем iSeries. 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.

Вызов команд

Класс CommandCall позволяет программе на Java вызывать команды iSeries в пакетном режиме. Результат выполнения команды помещается в список объектов AS400Message.

Для работы CommandCall необходимо следующее:

- Текст исполняемой команды
- Объект AS400, соответствующий системе, в которой будет запускаться команда

Команда может быть передана конструктору, методу setCommand() или run(). После выполнения команды метод getMessageList() позволяет получить все сообщения iSeries, отправленные командой.

При использовании класса CommandCall объект AS400 автоматически подключается к системе iSeries. Информация об управлении соединениями приведена в разделе управление соединениями.

Если программа на Java и сервер iSeries расположены в одной системе, то по умолчанию IBM Toolbox for Java проверяет, защищены ли нити команд в этой системе. В случае положительного результата команда

запускается в отдельной нити, а не в процессе. Для того чтобы отключить эту проверку, необходимо в явном виде задать защиту нитей команд с помощью метода setThreadSafe().

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

Ниже приведен пример запуска команды на сервере iSeries с помощью класса CommandCall:

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

                    // Создание объекта вызова команды.
                    // В этой программе команда будет
                    // указана позже. 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. 466 содержит пример запуска пользовательской команды.

Пул соединений

Пул соединений позволяет совместно использовать соединения и управлять наборами (пулами) соединений с сервером iSeries. Например, приложение может получить соединение из пула, воспользоваться им, а затем возвратить его обратно в пул для дальнейшего применения.

Класс AS400ConnectionPool управляет пулом объектов AS400. Класс AS400JDBCConnectionPool представляет пул объектов AS400JDBCConnections, которые могут использоваться программами на Java благодаря поддержке API JDBC 2.0 Package в IBM Toolbox for Java. 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. С помощью методов, унаследованных от ConnectionPool, возможна установка следующих свойств пула соединений:

- максимального числа соединений в пуле
- максимального времени жизни одного соединения
- максимального времени простоя соединения

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. Например, пул соединений можно создать путем заполнения пула активными (заранее установленными) соединениями. 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). Для указания службы воспользуйтесь константами из класса AS400 (FILE, PRINT, COMMAND и т.п.).

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.

Дополнительная информация об управлении соединением с сервером iSeries, открытым с помощью классов AS400ConnectionPool, приведена в разделе Управление соединениями.

Пример: Применение AS400ConnectionPool

“Пример: Применение AS400ConnectionPool” на стр. 468 содержит информацию о повторном использовании объектов AS400.

Область данных

Класс DataArea - это абстрактный класс, представляющий область данных на сервере iSeries. 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:

- Получать размер области данных
- Получать имя области данных
- Получать объект системы AS400, в которой находится область данных
- Обновлять атрибуты области данных
- Задавать систему, в которой находится область данных

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

CharacterDataArea

Класс CharacterDataArea представляет область данных сервера, в которой хранятся символьные данные. 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:

- Очищать область данных, заполняя ее пробелами.
- Создавать в системе области символьных данных со свойствами по умолчанию
- Создавать область логических данных с указанными атрибутами
- Удалять область данных из системы

- Получать путь к объекту интегрированной файловой системы, представленному областью данных.
- Считывать все данные, содержащиеся в области данных
- Считывать указанный объем данных с начала области данных или по заданному смещению
- Задавать полный путь к области данных в интегрированной файловой системе
- Записывать данные в начало области данных
- Записать указанный объем данных в начало области данных или по заданному смещению

DecimalDataArea

Класс DecimalDataArea представляет область десятичных данных сервера.

Класс DecimalDataArea позволяет выполнять следующие операции:
- Очищать область данных, заполняя ее нулями
- Создавать в системе области десятичных данных со свойствами по умолчанию.
- Создавать области десятичных данных с указанными атрибутами
- Удалять области данных с тех серверов, на которых они есть
- Получать число цифр после десятичной точки
- Получать путь к объекту интегрированной файловой системы, соответствующему области данных.
- Считывать все данные, содержащиеся в области данных
- Задавать полный путь к области данных в интегрированной файловой системе
- Записывать данные в начало области данных

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.

```
   // Подключение к серверу "My400".
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

Класс LocalDataArea представляет локальную область данных сервера. 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:

- Очищать область данных, заполняя ее пробелами.
- Считывать все данные, содержащиеся в области данных
- Считывать указанный объем данных с начала области данных или по заданному смещению
- Записывать данные в начало области данных
- Записать указанный объем данных в начало области данных или по заданному смещению

LogicalDataArea

Класс LogicalDataArea представляет область данных сервера, в которой хранятся логические данные.

Класс LogicalDataArea позволяет выполнять следующие операции:

- Очищать область данных, заполняя ее значениями false.
- Создавать на сервере области логических данных со свойствами по умолчанию
- Создавать области логических данных с указанными атрибутами
- Удалять области данных с серверов, на которых они есть
- Получать путь к объекту интегрированной файловой системы, соответствующему области данных
- Считывать все данные, содержащиеся в области данных
- Задавать полный путь к области данных в интегрированной файловой системе
- Записывать данные в начало области данных

DataAreaEvent

Класс DataAreaEvent описывает событие, относящееся к области данных.

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

- Получить идентификатор события

DataAreaListener

Класс DataAreaListener предоставляет интерфейс для получения сообщений о событиях области данных.

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

- Очистить
- Создание
- Удаление
- Чтение
- Запись

Преобразование и описание данных

Классы **преобразования данных** обеспечивают преобразование числовых и символьных данных между форматами iSeries и Java. Такое преобразование выполняется при обращении к данным iSeries из программы на Java. Предусмотрены классы для преобразования между различными числовыми форматами и между кодовыми страницами EBCDIC и Unicode.

Классы **описания данных** построены на базе классов преобразования данных и предназначены для преобразования всех полей записи путем вызова одного метода. Класс RecordFormat позволяет задать описание данных, хранящихся в параметрах DataQueueEntry и ProgramCall, описание записи файла базы

данных, для работы с которым применяются классы доступа на уровне записи, а также описание буфера данных iSeries. Класс Record позволяет преобразовывать содержимое всей записи и обращаться к отдельным полям записи по имени или индексу поля.

Классы **преобразования** обеспечивают быстрое и эффективное двустороннее преобразование объектов Java и iSeries. Класс BinaryConverter преобразует массивы байтов Java в простые типы Java и обратно. Класс CharConverter преобразует объекты Java String в кодовые страницы OS/400 и обратно. Дополнительная информация приведена в следующем разделе:

Классы преобразования

Типы данных

AS400DataType - это интерфейс с набором методов преобразования данных. Эти методы реализуются 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

IBM Toolbox for Java позволяет создавать классы для преобразования целых записей, а не отдельных полей данных. Например, пусть программа на Java получает данные из очереди данных. При этом из очереди считывается массив двоичных данных iSeries. Этот массив может содержать данные iSeries различных типов. Приложение, вместо того чтобы преобразовывать эти данные по одному полю, может создать формат записи, описывающий поля массива. Этот формат будет обеспечивать преобразование всех данных записи 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. Программы преобразования формата записи упрощают преобразование данных между форматами iSeries и Java.

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

Классы преобразования для числовых данных:

Классы преобразования числовых данных преобразуют числовые данные из формата, применяемого на сервере iSeries (называемого в приведенной ниже таблице **форматом сервера**), в формат данных Java. Поддерживаемые типы перечислены в следующей таблице:

Числовой тип	Описание
AS400Bin2	Преобразует двухбайтовое число со знаком в объект Short языка Java (и наоборот).
AS400Bin4	Преобразует четырехбайтовое число со знаком в объект Integer языка Java (и наоборот).
AS400ByteArray	Преобразует один массив байт в другой. Этот преобразование применяется для правильного дополнения целевого буфера нулями.
AS400Float4	Преобразует четырехбайтовое число с плавающей точкой и со знаком в объект Float языка Java (и наоборот).
AS400Float8	Преобразует восьмибайтовое число с плавающей точкой и со знаком в объект Double языка Java (и наоборот).
AS400PackedDecimal	Преобразует упакованное десятичное число в объект BigDecimal языка Java (и наоборот).
AS400UnsignedBin2	Преобразует двухбайтовое число без знака в объект Integer языка Java (и наоборот).
AS400UnsignedBin4	Преобразует четырехбайтовое число без знака в объект Long языка Java (и наоборот).
AS400ZonedDecimal	Преобразует зонное десятичное число в объект BigDecimal языка Java (и наоборот).

Примеры

Ниже приведен пример преобразования числового типа сервера в тип int языка Java:

Пример: Преобразование формата сервера в формат int языка Java

```
// Создание буфера для размещения данных сервера. Предполагается, что в буфере
// содержатся такие данные числового формата сервера, как записи очередей данных,
// вызовы программ и т.д.
byte[] data = new byte[100];

// Создание объекта преобразования для этого типа данных сервера.
AS400Bin4 bin4Converter = new AS400Bin4();

// Преобразование данных сервера в объект Java. Число расположено в начале
// буфера.
Integer intObject = (Integer) bin4Converter.toObject(data,0);

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

Пример: Преобразование данных типа int языка Java в формат сервера

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

// Создание объекта преобразования для этого типа данных сервера.
AS400Bin4 bin4Converter = new AS400Bin4();
```

```
// Преобразование объекта Java в данные сервера.
byte[] data = bin4Converter.toBytes(intObject);

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

Преобразование текста:

Класс AS400Text предназначен для преобразования символьных данных. Этот класс преобразует текст между кодовой страницей EBCDIC с заданным CCSID и кодировкой Unicode. При создании объекта AS400Text указывается длина преобразуемой строки и CCSID или кодировка данных сервера. Предполагается, что в программе на Java используется CCSID 13488 Unicode. Метод toBytes() преобразует формы Java в массивы байтов в формате iSeries. Метод toObject() преобразует данные iSeries из массива байтов в объект Java.

Класс AS400BidiTransform позволяет преобразовать двунаправленный текст из формата iSeries в формат Java (с промежуточным преобразованием в Unicode) или наоборот. По умолчанию преобразование выполняется с учетом CCSID задания. Для изменения направления ввода текста и способа начертания символов укажите атрибут BidiStringType. Обратите внимание, что в объектах IBM Toolbox for Java, выполняющих неявное преобразование данных, например, в объекте DataArea, предусмотрен метод для переопределения строкового типа. 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:

```
// Предполагается, что записи очереди данных уже извлечены из системы
// iSeries и помещены
// в следующий буфер.
int textLength = 100;
byte[] data = new byte[textLength];

// Создание объекта для преобразования типа данных iSeries. Создается
// объект-преобразователь по умолчанию. Предполагается, что кодовая
// страница EBCDIC на сервере iSeries совпадает с локалью клиента. В
// противном случае, в программе можно явно указать CCSID EBCDIC.
// Рекомендуется по возможности всегда указывать CCSID.
// (см. Примечания).
AS400Text textConverter = new AS400Text(textLength)

// Примечание: Объект-преобразователь можно создать для конкретного
// CCSID. Если программа работает как клиент proxy Toolbox for Java,
// необходимо использовать объект AS400.
int ccsid = 37;
AS400 system = ...; // Объект AS400
AS400Text textConverter = new AS400Text(textLength, ccsid, system);

// Примечание: Можно создать объект-преобразователь с помощью одного
// объекта AS400. Предполагается, что кодовая страница iSeries
// совпадает с CCSID, возвращенным объектом AS400.
AS400Text textConverter = new AS400Text(textLength, system);

// Преобразование данных из формата EBCDIC в Unicode. Если длина
// объекта AS400Text превышает число преобразуемых
// символов, полученный объект String будет
// дополнен пробелами до указанной длины.
String javaText = (String) textConverter.toObject(data);
```

Классы преобразования для составных типов:

Преобразование составных типов обеспечивается следующими классами:

- AS400Array - Позволяет программе на Java работать с массивами однотипных данных.
- AS400Structure - Позволяет программе на Java работать со структурами разнотипных элементов.

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.

```
// Создание структуры типов данных, соответствующей структуре, которая
// содержит: - четырехбайтовый номер
//    - четыре байта выравнивания
//    - восьмибайтовое число
//    - 40 символов
AS400DataType[] myStruct =
{
   new AS400Bin4(),
   new AS400ByteArray(4),
   new AS400Float8(),
   new AS400Text(40)
};

// Создание объекта преобразования с помощью структуры.
AS400Structure myConverter = new AS400Structure(myStruct);

// Создание объекта Java, содержащего данные для отправки на сервер.
Object[] myData =
{
   new Integer(88),        // четырехбайтовое число
   new byte[0],            // выравнивание (здесь - нулевого размера)
   new Double(23.45),      // восьмибайтовое действительное число
   "This is my structure"  // строка символов
};

// Преобразование объекта Java в двоичный массив.
byte[] myAS400Data = myConverter.toBytes(myData);


// ... отправка массива байтов на сервер. Получение данных с
// сервера. Полученные данные также будут массивом байтов.


// Преобразование полученных из iSeries данных в формат Java.
Object[] myRoundTripData = (Object[])myConverter.toObject(myAS400Data,0);

// Получение третьего объекта структуры. Объект имеет тип double.
Double doubleObject = (Double) myRoundTripData[2];

// Извлечение простого типа Java из объекта Java.
double d = doubleObject.doubleValue();
```

Классы FieldDescription:

Классы описания полей позволяют программам на Java задавать описания полей, содержащие тип данных и имя поля. При работе программы с данными на уровне записей они могут также указывать ключевые слова Спецификации описания данных (DDS) iSeries, содержащие дополнительную информацию о поле.

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:

Класс RecordFormat позволяет программе на Java описать группу полей или параметров. 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:

- Добавление описания поля к формату записи.
- Добавление описания ключевого поля к формату записи.
- Получение описания поля из формата записи по индексу или имени поля.
- Получение описания ключевого поля из формата записи по индексу или имени поля.
- Получение имен полей, описанных в формате записи.
- Получение имен ключевых полей, описанных в формате записи.
- Получение числа полей, описанных в формате записи.
- Получение числа ключевых полей, описанных в формате записи.
- Создание записи с заданным форматом.

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"

Класс Record:

Класс записи позволяет программам на Java работать с данными, формат которых описан в классе формата записи. 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");
```

Получение содержимого поля:

Может потребоваться получить как одно поле объекта Record, так и все его поля сразу. Для получения содержимого поля по его имени или индексу предназначен метод getField(). Для получения содержимого всех полей в виде массива объектов предназначен метод getFields().

Полученный объект (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

DDS поля	Значение FieldDescription поля	Объект Java
Двоичные данные (B), длина >= 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

Задание содержимого поля:

В программе на Java содержимое объекта Record можно задать с помощью метода setField(). Значение поля 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
Двоичные данные (B), длина >= 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

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.

Для применения LineDataRecordWriter необходимо задать следующие атрибуты формата записи:

- ИД формата записи
- Тип формата записи

Применение классов Record и RecordFormat позволяет LineDataRecordWriter принимать запись в качестве ввода для метода writeRecord(). (При создании экземпляра записи указывается RecordFormat.)

Класс LineDataRecordWriter содержит методы, позволяющие:

- Узнать текущий CCSID
- Получить имя кодировки
- Поместить запись в строковом формате в 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.");
}
```

Очереди данных

Классы DataQueue позволяют программам на Java работать с очередями данных сервера. Ниже перечислены некоторые свойства очередей данных системы iSeries:

- 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.

В качестве обязательного параметра любого объекта очереди данных указывается объект AS400, представляющий сервер, на котором находится или будет создана очередь данных.

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

Классы очередей данных поддерживают очереди данных следующих типов:

- Последовательные очереди данных
- Очереди данных с извлечением по ключу

Методы, общие для всех очередей данных, находятся в классе BaseDataQueue. Класс DataQueue расширяет класс BaseDataQueue за счет методов, поддерживающих очереди данных с последовательным извлечением. Класс KeyedDataQueue расширяет класс BaseDataQueue за счет методов, необходимых для работы с очередями данных с извлечением по ключу.

Данные, считанные из очереди, возвращаются в виде объекта DataQueueEntry. В этом объекте могут храниться данные из очередей обоих типов. Для получения дополнительной информации из очереди с извлечением по ключу применяется объект класса KeyedDataQueueEntry, расширяющего класс DataQueueEntry.

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);
```

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). Классы BaseDataQueue и DataQueue содержат следующие методы для работы с очередями данных с последовательным извлечением:

- Создание очереди данных на сервере. Необходимо указать максимальный размер передаваемой записи. Кроме того, при создании можно указать набор дополнительных параметров (FIFO или LIFO, следует ли сохранять информацию об отправителе, следует ли сохранять на диске, описание очереди).
- Чтение записи из очереди без удаления ее из очереди. При отсутствии данных выполнение программы может быть приостановлено до поступления нужной записи или продолжено.
- Чтение записи с удалением ее из очереди. При отсутствии данных выполнение программы может быть приостановлено до поступления нужной записи или продолжено.
- Добавление записи к очереди данных.
- Очистка очереди с удалением всех записей.
- Удаление очереди данных.

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. 473

“Пример: Применение классов DataQueue для считывания записей из очереди данных” на стр. 476

Очереди данных с извлечением по ключу:

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

- Создание очереди данных с извлечением по ключу на сервере. Необходимо указать размер ключа и максимальный размер передаваемой записи. Кроме того, при создании можно указать набор дополнительных параметров (права доступа, следует ли сохранять информацию об отправителе, следует ли сохранять на диске, описание очереди).
- Чтение записи с указанным ключом из очереди без удаления ее из очереди. При отсутствии записи с указанным ключом выполнение программы может быть приостановлено до поступления нужной записи или продолжено.
- Чтение записи с указанным ключом с удалением ее из очереди. При отсутствии записи с указанным ключом выполнение программы может быть приостановлено до поступления нужной записи или продолжено.
- Добавление записи с ключом в очередь данных.
- Очистка очереди с удалением всех записей или записей с определенным ключом.
- Удаление очереди данных.

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. 481

“Пример: Применение классов KeyedDataQueue для считывания записей из очереди данных” на стр. 484

Цифровые сертификаты

Цифровые сертификаты - это документы с цифровой подписью, используемые для защиты транзакций в Internet. (Цифровые сертификаты поддерживаются на серверах с операционной системой OS/400 версии 4, выпуска 3 (V4R3) и более поздних версий). 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.

Для создания цифровых сертификатов можно использовать инструмент javakey. (Дополнительная информация о javakey и средствах защиты Java приведена на Web-странице Sun Microsystems, Inc., Java

Security  .) В лицензионной программе IBM Toolbox for Java предусмотрены классы для работы с цифровыми сертификатами на серверах iSeries и AS/400e.

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.

- Класс AS400CertificateUserProfileUtil содержит методы управления сертификатами пользовательских профайлов.
- 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. Класс AS400CertificateAttribute содержит атрибуты, которые могут применяться в качестве критериев выбора сертификатов. 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 они помещаются в пользовательское пространство. Для этого применяется метод listCertificates().
5. С помощью метода getCertificates() можно получать сертификаты из пользовательского пространства.

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
```

Класс EnvironmentVariable

Классы EnvironmentVariable и EnvironmentVariableList позволяют получать и задавать **системные** переменные среды iSeries.

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:

- Получать и задавать имя
- Получать и задавать систему
- Получать и задавать значение переменной (в частности, изменять CCSID)
- Обновлять значение переменной

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.

```
// Создание объекта, представляющего iSeries.
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();
```

Исключительные ситуации

Классы доступа IBM Toolbox для Java генерируют исключительные ситуации при возникновении ошибок устройств и достижении физических ограничений, а также при программных ошибках и ошибках при вводе данных. 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);
   }
}
```

Класс FTP

Класс FTP предоставляет программный интерфейс для работы с функциями FTP. 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:

- Подключаться к серверу FTP
- Передавать команды на сервер
- Получать список файлов в каталоге
- Получать файлы с сервера **и**
- Помещать файлы на сервер

Пример: Копирование файлов с сервера с помощью FTP

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

Например, с помощью класса FTP можно скопировать набор файлов из каталога на сервере:

```
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

Если класс FTP предоставляет общий интерфейс для работы с FTP, то подкласс AS400FTP специально предназначен для работы с сервером FTP в системе. Этот класс разработан в соответствии с семантикой сервера FTP систем iSeries. Например, этот класс содержит функции, применяемые при передаче файла сохранения на сервер, что позволяет выполнять эту операцию автоматически. AS400FTP также связан со средствами защиты IBM Toolbox for Java. Как и другие классы IBM Toolbox for Java, AS400FTP применяет объект AS400 для работы с именем системы, идентификатором пользователя и паролем.

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");
```

Интегрированная файловая система

Классы интегрированной файловой системы позволяют программам на Java работать с файлами интегрированной файловой системы сервера iSeries как с потоком байтов или потоком символов. Эти классы потребовалось создать в силу того, что пакет java.io не поддерживает перенаправления ввода-вывода в файл и других функций системы iSeries.

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)

С помощью классов интегрированной файловой системы программа на Java может напрямую работать с потоковыми файлами системы iSeries. Программа на Java может по-прежнему использовать пакет java.io, однако в этом случае клиентская операционная система должна поддерживать метод перенаправления. Например, если программа на Java выполняется в операционной системе Windows 95 или Windows NT, то для перенаправления вызовов java.io на сервер iSeries требуется функция для работы с сетевыми дисками продукта iSeries Access для Windows. При работе с классами интегрированной файловой системы продукт iSeries Access для Windows не требуется.

Обязательным параметром любого класса интегрированной файловой системы является объект AS400, представляющий систему iSeries, в которой расположен файл. При использовании классов интегрированной файловой системы объект AS400 автоматически подключается к системе iSeries. 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
```

Классы IFS

В следующей таблице перечислены классы интегрированной файловой системы.

Класс интегрированной файловой системы	Описание
IFSFile	Соответствует файлу в интегрированной файловой системе
IFSJavaFile	Соответствует файлу в интегрированной файловой системе (расширение java.io.File)
IFSFileInputStream	Представляет поток ввода для чтения данных из файла системы iSeries
IFSTextFileInputStream	Соответствует потоку символьных данных, считываемых из файла
IFSFileOutputStream	Представляет поток вывода для записи данных в файл системы iSeries
IFSTextFileOutputStream	Соответствует потоку символьных данных, записываемых в файл
IFSRandomAccessFile	Представляет файл системы iSeries, применяемый для чтения или записи данных
IFSFileDialog	Предоставляет пользовательский интерфейс для перемещения по структуре каталогов и выбора файла

Примеры: Применение классов IFS

“Пример: Применение классов IFS для копирования файла из одного каталога в другой” на стр. 492 содержит пример копирования файла из одного каталога iSeries в другой с помощью классов интегрированной файловой системы.

“Пример: Применение классов IFS для просмотра содержимого каталога” на стр. 494 приведен пример того, как с помощью классов IFS можно просмотреть список объектов каталога iSeries.

Класс IFSFile:

Класс IFSFile представляет объект интегрированной файловой системы iSeries. 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:

- Определять, существует ли объект, и является ли он каталогом или файлом
- Определять, есть ли у программы на Java права на чтение файла или запись в файл
- Узнавать размер файла
- Получать права доступа к объекту и задавать их
- Создавать каталоги
- Удалять файлы и каталоги
- Переименовывать файлы и каталоги
- Получать и задавать дату последнего изменения файла
- Считывать содержимое каталогов
- Просматривать содержимое каталога и сохранять атрибуты в локальном кэше
- Определять объем свободной памяти системы
- Определять тип объекта, расположенного в файловой системе QSYS.LIB

Список файлов каталога можно просмотреть с помощью метода list() или метода 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. 487
- “Пример: Применение исключительных ситуаций IFSFile для отслеживания ошибок” на стр. 488
- “Пример: Просмотр файлов с расширением .txt” на стр. 489
- “Пример: Применение метода listFiles() класса IFSFile для просмотра содержимого каталога” на стр. 489

Класс IFSJavaFile:

Класс IFSJavaFile представляет файл интегрированной файловой системы iSeries. Он расширяет класс java.io.File. IFSJavaFile позволяет создавать совместимые с интерфейсом java.io.File программы, которые работают с интегрированной файловой системой iSeries.

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. После этого вся информация о файлах берется из кэша.
- Метод list() получает информацию о каждом файле путем обращения к серверу, поэтому он работает медленнее и сильнее зависит от производительности сервера.

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());
}
```

IFSFileInputStream:

Класс IFSFileInputStream представляет поток ввода для чтения данных из файла на сервере. Как и в классе IFSFile, в IFSFileInputStream присутствуют методы, дублирующие методы класса FileInputStream из пакета java.io. Помимо них в классе IFSFileInputStream есть методы, относящиеся непосредственно к серверам iSeries. Класс IFSFileInputStream позволяет программе на Java выполнять следующие операции:

- Открывать файл для чтения. Файл должен существовать, так как этот класс не создает файлов на сервере. В конструкторе класса можно задать режим использования файла.
- Определять число байт в потоке.
- Считывать определенное количество байт данных из потока.
- Пропускать заданное количество байт в потоке.
- Блокировать и разблокировать байты в потоке.
- Закрывать 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();
```

Класс IFSTextFileInputStream:

Класс IFSTextFileInputStream представляет поток символьных данных, считываемых из файла. Данные, считанные из объекта IFSTextFileInputStream, передаются программе на Java в виде объекта String, 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. 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();
```

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:

- Открывать файл для записи. Если файл существует, он заменяется. С помощью конструкторов класса можно задать режим использования файла и указать, нужно ли сохранять содержимое существующего файла.
- Записывать заданное число байт данных в поток.
- Фиксировать на диске байты данных, записанные в поток.
- Блокировать и разблокировать байты в потоке.
- Закрывать 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();
```

Класс IFSTextFileOutputStream:

Класс IFSTextFileOutputStream представляет поток символьных данных, записываемых в файл. Объекту IFSTextFileOutputStream передаются данные в виде объекта String, 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();
```

IFSRandomAccessFile:

Класс IFSRandomAccessFile представляет файл сервера, открытый для чтения или записи. 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. Помимо этих методов, класс IFSRandomAccessFile содержит специальные методы для применения на сервере iSeries. С помощью класса IFSRandomAccessFile программа на Java может выполнять следующие действия:

- Открывать файл для чтения, записи, а также чтения и записи. При открытии программа может задать режим использования файла и опцию открытия/создания файла.
- Считывать данные из файла, начиная с текущего смещения.
- Записывать данные в файл, начиная с текущего смещения.
- Получать и задавать текущее смещение в файле.
- Закрывать файл.

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

- Фиксировать на диске записанные данные.
- Блокировать и разблокировать данные в файле.
- 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();
```

IFSFileDialog:

Класс IFSFileDialog позволяет просматривать содержимое файловой системы и выбирать файлы. Этот класс применяет класс IFSFile для чтения списка каталогов и файлов интегрированной файловой системы сервера iSeries. Методы класса позволяют программам на Java задавать текст кнопок в окне диалога и устанавливать фильтры. Обратите внимание, что существует версия IFSFileDialog, использующая Swing 1.1.

С помощью класса FileFilter можно задать фильтры. Имя файла, выбранного в окне диалога, можно получить с помощью метода getFileName(). Полное имя выбранного файла можно получить с помощью метода getAbsolutePath().

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
                    // установка фильтров в окне диалога.
                    // 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());
```

Класс IFSKey:

Когда программа на Java работает с файлом, доступным другим программам, она может заблокировать байтовый фрагмент в этом файле на определенное время. В течение этого времени программа будет обладать исключительным доступом к фрагменту. После получения блокировки класс интегрированной файловой системы возвращает объект IFSKey. 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");

                    // Открытие потока ввода. Конструктор
                    // 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.
                    //  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();
```

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();
```

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. Задать переменную среды CLASSPATH на сервере с помощью метода setClassPath()
2. Определить параметры программы с помощью метода setParameters()
3. Запустить программу с помощью метода run()
4. Передать введенные данные из клиентской системы программе на Java. Программа считывает данные через стандартный поток ввода, который задается методом sendStandardInString(). Стандартные потоки вывода и ошибок могут быть перенаправлены в клиентскую систему с помощью методов getStandardOutString() и getStandardErrorString()

Класс JavaApplicationCall - это класс, вызываемый из программы на Java. Однако в 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

Классы JDBC

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

Дополнительная информация о JDBC и поддержке JDBC в IBM Toolbox for Java, включая список планируемых улучшений, свойства JDBC и не поддерживаемые типы SQL, приведены на следующей странице:

“JDBC” на стр. 322

Поддерживаемые интерфейсы

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

Поддерживаемый интерфейс JDBC	**Необходимый API**
Интерфейс Blob предназначен для работы с большими двоичными объектами (BLOB).	Базовый API JDBC 2.1
CallableStatement запускает хранимые процедуры SQL	JDK 1.1
Интерфейс Clob предназначен для работы с большими символьными объектами (CLOB).	Базовый API JDBC 2.1
Интерфейс Connection представляет соединение с некоторой базой данных.	JDK 1.1
Объект ConnectionPool представляет пул объектов соединений.	Дополнительный пакет JDBC 2.0
Объект ConnectionPoolDataSource представляет фабрику классов, помещенных в пул объектов AS400JDBCPooledConnection	Дополнительный пакет JDBC 2.0
Объект DatabaseMetaData предоставляет информацию обо всей базе данных в целом.	JDK 1.1
Объект DataSource представляет фабрику соединений с базой данных.	Дополнительный пакет JDBC 2.0

Поддерживаемый интерфейс JDBC	Необходимый API
Интерфейс Driver устанавливает соединение и возвращает версию драйвера.	JDK 1.1
Интерфейс ParameterMetaData позволяет получать информацию о типах и свойствах параметров объектов PreparedStatement	API JDBC 3.0
Интерфейс PreparedStatement выполняет откомпилированные операторы SQL	JDK 1.1
Интерфейс ResultSet предназначен для работы с таблицей, полученной в результате выполнения запроса SQL или метода из каталога DatabaseMetaData.	JDK 1.1
Интерфейс ResultSetMetaData возвращает информацию о некотором наборе результатов	JDK 1.1
RowSet - это набор строк, включающий в себя объект ResultSet	Дополнительный пакет JDBC 2.0
Интерфейс Savepoint содержит средства для более точного управления транзакциями	API JDBC 3.0
Интерфейс Statement запускает оператор SQL и возвращает результат его выполнения.	JDK 1.1
XAConnection представляет соединение с базой данных, которое применяется для выполнения глобальных транзакций XA	Дополнительный пакет JDBC 2.0
Интерфейс XAResource - диспетчер ресурсов, применяемый для выполнения транзакций XA	Дополнительный пакет JDBC 2.0

Примеры

В приведенных ниже примерах продемонстрированы различные способы применения драйвера JDBC IBM Toolbox for Java.

- “Пример: Применение класса JDBCPopulate для создания и заполнения таблицы” на стр. 497
- “Пример: Применение класса JDBCQuery для отправки запроса к таблице” на стр. 499

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:

- Получить весь объект blob в виде потока непреобразованных байтов

- Получить часть содержимого объекта blob
- Получить размер объекта blob
- Создать двоичный поток для записи в объект blob
- Сохранить массив байтов в объекте blob
- Полностью или частично сохранить объект байтов в объекте blob
- Усечь объект 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

С помощью объекта AS400JDBCBlobLocator можно работать с большими двоичными объектами (blob).

Класс AS400JDBCBlobLocator позволяет выполнять следующие операции:

- Получить весь объект blob в виде потока непреобразованных байтов
- Получить часть содержимого объекта blob
- Получить размер объекта blob
- Создать двоичный поток для записи в объект blob
- Сохранить массив байтов в объекте blob
- Полностью или частично сохранить объект байтов в объекте blob
- Усечь объект blob

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();
```

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. Информация о других дополнительных свойствах приведена в разделе “IBM Toolbox for Java - Свойства JDBC” на стр. 327.

Класс AS400JDBCClob позволяет выполнять следующие операции:

- Получить весь объект clob в виде потока символов ASCII
- Получить содержимое объекта clob в виде потока символов
- Получить часть содержимого объекта clob
- Возвратить длину объекта clob
- Создать поток символов Unicode или поток символов ASCII для записи в объект clob
- Сохранить строку в объекте clob
- Усечь объект 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

Объект AS400JDBCClobLocator предназначен для работы с большими символьными объектами (CLOB).

Класс AS400JDBCClobLocator позволяет выполнять следующие операции:

- Получить весь объект clob в виде потока символов ASCII
- Получить весь объект clob в виде потока символов
- Получить часть содержимого объекта clob
- Возвратить длину объекта clob
- Создать поток символов Unicode или поток символов ASCII для записи в объект clob
- Сохранить строку в объекте clob
- Усечь объект clob

Класс AS400JDBCConnection:

Класс AS400JDBCConnection представляет соединение JDBC с базой данных DB2 UDB for iSeries. Для создания объектов AS400JDBCConnection предназначен метод DriverManager.getConnection(). Дополнительная информация приведена в разделе “Регистрация драйвера JDBC” на стр. 71.

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 - Свойства JDBC” на стр. 327 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 “Изменения, внесенные в поддержку JDBC продукта IBM Toolbox for Java” на стр. 323.

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. Дополнительная информация приведена в разделе “Класс AS400” на стр. 27 и Документация по защите J2SDK версии 1.4  .

Класс AS400JDBCConnection позволяет выполнять следующие операции:

- Создавать операторы (объекты Statement, PreparedStatement и CallableStatement)
- Создавать операторы с указанным типом набора результатов и уровнем распараллеливания (объекты Statement, PreparedStatement и CallableStatement)
- Выполнять фиксацию и откат изменений, внесенных в базу данных, а также разблокировать объекты базы данных
- Закрывать соединения с немедленным освобождением ресурсов сервера (вместо ожидания автоматического освобождения)
- Задавать и получать уровень блокировки, установленный для соединения
- Задавать и получать уровень изоляции транзакций, установленный для соединения
- Получать мета-данные соединения
- Включать и выключать режим автоматической фиксации
- Получать идентификатор задания сервера хоста, связанный с соединением

Если применяется JDBC 3.0 и объект AS400JDBCConnection применяется для подключения к серверу с операционной системой OS/400 выпуска V5R2 или выше, то с его помощью можно выполнять следующие действия:

- Создавать операторы с определенным уровнем блокировки набора результатов (объекты Statement, PreparedStatement и CallableStatement)
- Создавать подготовленный оператор, который будет возвращать все автоматически созданные ключи (при вызове метода getGeneratedKeys() в объекте оператора)
- Применять точки сохранения, предоставляющие гибкие средства управления транзакциями:
  - Задавать точки сохранения
  - Выполнять откат до точки сохранения
  - Разблокировать точки сохранения

AS400JDBCConnectionPool:

Класс AS400JDBCConnectionPool представляет пул объектов AS400JDBCConnection, которые могут быть использованы в программах на Java. Этот класс входит в состав поддержки IBM Toolbox for Java для API Дополнительного пакета JDBC 2.0.

Класс AS400JDBCConnectionPoolDataSource позволяет задать свойства соединений, создаваемых в пуле, как показано в следующем примере.

Источник данных пула соединений нельзя изменить после того, как был сделан запрос на соединение и пул был занят системой. Для изменения источника данных нужно вызвать метод close() для этого пула.

Для возвращения соединений в пул AS400JDBCConnectionPool вызовите метод close() для объекта AS400JDBCConnection.

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

Для настройки свойств пула служат методы, унаследованные от класса ConnectionPool. Ниже перечислены некоторые из этих свойств:

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

Вы можете зарегистрировать объекты AS400JDBCConnectionPoolDataSource с помощью поставщика услуг Java Naming and Directory Interface(TM) (JNDI). Более подробную информацию о поставщиках услуг 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();
```

Интерфейс DatabaseMetaData:

Объект DatabaseMetaData предназначен для получения информации о всей базе данных и о каталогах.

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();
```

Класс AS400JDBCDataSource:

Класс AS400JDBCDataSource представляет фабрику классов для соединений с базой данных iSeries. Класс AS400JDBCConnectionPoolDataSource представляет фабрику классов для объектов AS400JDBCPooledConnection.

Для регистрации объектов источников данных обоих типов служит поставщик услуг Java Naming and Directory Interface (JNDI). Более подробную информацию о поставщиках услуг JNDI можно найти в разделе IBM Toolbox for Java - Ссылки на справочную информацию.

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");

// Создание соединения с базой данных iSeries.
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:

```
// Создание источника данных для базы данных iSeries.
AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
dataSource.setServerName("myAS400");
dataSource.setDatabaseName("myAS400 Database");

// Регистрация источника данных с помощью JNDI.
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.fscontext.RefFSContextFactory");
Context context = new InitialContext(env);
context.bind("jdbc/customer", dataSource);

// Получение от JNDI объекта AS400JDBCDataSource и создание соединения.
AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup("jdbc/customer");
Connection connection = datasource.getConnection("myUser", "MYPWD");
```

Пример: Использование объектов AS400JDBCDataSource и классов IBM SecureWay Directory на сервере простого протокола доступа к каталогам (LDAP)

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

```
// Создание источника данных для базы данных iSeries.
AS400JDBCDataSource dataSource = new AS400JDBCDataSource();
dataSource.setServerName("myAS400");
dataSource.setDatabaseName("myAS400 Database");

// Регистрация источника данных с помощью JNDI.
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.jndi.LDAPCtxFactory");
Context context = new InitialContext(env);
context.bind("cn=myDatasource, cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion",
             dataSource);

// Получение от JNDI объекта AS400JDBCDataSource и создание соединения.
AS400JDBCDataSource datasource = (AS400JDBCDataSource) context.lookup(
   "cn=myDatasource, cn=myUsers, ou=myLocation,o=myCompany,c=myCountryRegion");
Connection connection = datasource.getConnection("myUser", "MYPWD");
```

Регистрация драйвера JDBC:

Перед получением информации из базы данных сервера с помощью драйвера JDBC, его нужно зарегистрировать в лицензионной программе IBM Toolbox for Java с помощью класса 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

  Для того чтобы загрузить драйвер JDBC Toolbox for Java, добавьте в программу на Java перед первым вызовом JDBC следующий оператор:

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

Драйвер JDBC Toolbox for Java автоматически зарегистрируется при загрузке. Такой способ регистрации драйвера является предпочтительным. Для того чтобы явно зарегистрировать драйвер JDBC, укажите следующий оператор:

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

В отличие от остальных классов IBM Toolbox for Java, получающих информацию с сервера, драйвер JDBC не требует передачи объекта AS400 в качестве входного параметра. 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. Как и в других функциях IBM Toolbox for Java, если ИД пользователя и пароль предоставляются программой на Java, ИД пользователя не запоминается как ИД по умолчанию, а пароль не заносится в кэш. Информация об управлении соединениями приведена в разделе “Управление соединениями” на стр. 444.

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. При работе с драйвером IBM Toolbox for Java URL должен быть задан в следующем формате:

```
"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. Дополнительная информация приведена в разделе “Класс AS400” на стр. 27 и Документация по защите J2SDK версии 1.4 .

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

```
                    // Подключение к системе 'mySystem'.
                    // Не задана схема по умолчанию и
                    // свойства системы.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem");
```

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

```
                    // Подключение к системе '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 - Свойства JDBC” на стр. 327.

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
                    // свойств.
Connection c = DriverManager.getConnection("jdbc:as400://mySystem",p);
```

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

```
                    // Подключение к системе.
                    // 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();
```

Класс AS400JDBCParameterMetaData:

Класс AS400JDBCParameterMetaData позволяет программам получать информацию о свойствах параметров объектов PreparedStatement и CallableStatement.

Методы класса AS400JDBCParameterMetaData позволяют выполнять следующие операции:

- Получать имя класса параметра
- Получать число параметров объекта PreparedStatement
- Получать тип SQL параметра
- Получать тип параметра, назначенный ему в базе данных
- Получать точность или длину дробной части параметра

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);
```

Интерфейс PreparedStatement:

Объект PreparedStatement применяется в тех случаях, когда оператор SQL должен выполняться многократно. Такой оператор можно откомпилировать заранее. В этом случае он будет называться ″подготовленным″ оператором SQL. 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
                  // операторов SQL. Вопросительным знаком
                  // 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();
```

Класс ResultSet:

Объект ResultSet предназначен для работы с таблицей данных, полученной в результате обработки запроса.Строки таблицы просматриваются последовательно. Поля одной строки можно просматривать в любом порядке.

Для получения данных из объекта ResultSet применяются методы get для соответствующих типов данных.Для перехода к следующей строке применяется метод next().

ResultSet позволяет получать и обновлять столбцы по именам; тем не менее, выполнение этой операции с помощью индексов занимает меньше времени.

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	isFirst
afterLast	isLast
beforeFirst	last
first	moveToCurrentRow
getRow	moveToInsertRow
isAfterLast	previous
isBeforeFirst	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.

Примечание: IBM iSeries Server поддерживает работу с таблицами без учета изменений в режиме только для чтения. IBM Toolbox for Java поддерживает работу с таблицами, доступными только для чтения, только в режиме без учета изменений. Если таблица результатов работает в режиме без учета изменений и режим можно изменить, таблица переводится в режим работы с учетом изменений и пользователю отправляется 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:

- Обновление потока символов ASCII
- Обновление большого десятичного числа
- Обновление потока двоичных данных

Полный список методов обновления интерфейса ResultSet приведен в разделе Обзор методов.

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

Интерфейс ResultSetMetaData задает типы и свойства столбцов таблицы результатов.

При подключении к серверу, работающему под управлением системы OS/400 версии V5R2 или выше, с помощью свойства расширенных метаданных можно повысить точность работы следующих методов ResultSetMetaData:

- 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.

Класс AS400JDBCRowSet:

Класс AS400JDBCRowSet представляет набор связанных строк, содержащий таблицу результатов JDBC. Методы класса AS400JDBCRowSet во многом схожи с методами класса 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();
```

Класс AS400JDBCSavepoint:

Класс AS400JDBCSavepoint представляет логическую точку прерывания транзакции. 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

В IBM Toolbox for Java применяются правила работы с базами данных, определяющие способы обработки курсоров полных блокировок при откатах. Например, если в параметрах соединения указано, что после обычного отката курсоры остаются открытыми, то то же самое произойдет и после отката к точке сохранения. Другими словами, при выполнении отката до точки сохранения IBM Toolbox for Java не перемещает и не закрывает курсоры, если такие операции не поддерживаются базой данных.

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.
- На сервере должна быть установлена операционная система OS/400 версии V5R2 или выше. Если соединение установлено с сервером, на котором установлена операционная система OS/400 версии V5R1 или ниже, то попытка создать точку сохранения приведет к возникновению исключительной ситуации.

Подробное описание рисунка 1: Применение точек сохранения для управления откатами транзакций (rzahh586.gif):

IBM Toolbox for Java: Класс AS400JDBCSavepoint

Этот рисунок иллюстрирует применение точек сохранения для управления откатами транзакций.

Описание

Рисунок состоит из следующих компонентов:

- Синей горизонтальной стрелки, указывающей слева направо, с надписью 'Транзакция'. Эта стрелка представляет линейную транзакцию, начинающуюся слева и заканчивающуюся справа.
- Под стрелкой Транзакция находится многоцветная полоска той же длины. Эта полоска разделена на четыре цветных участка, представляющих последовательные действия, составляющие транзакцию. Под полоской показаны две точки сохранения транзакции.
- Под многоцветной полоской находятся три стрелки. Каждая из стрелок (направленных справа налево) представляет откат транзакции до определенной точки.

Стрелка Транзакция представляет транзакцию, начинающуюся слева и заканчивающуюся справа. Транзакция состоит из последовательности отдельных действий (различные части многоцветной полоски). Слева направо участки полоски представляют:

- Первое действие (желтый участок) под названием 'Вставка'
- Второе действие (зеленый участок) под названием 'Обновление'
- Третье действие (желтый участок) под названием 'Вставка'
- Четвертое действие (синий участок) под названием 'Удаление'

Метки под многоцветной полоской представляют точки сохранения. Точка завершения первого действия и начала второго отмечена как 'Точка сохранения A'. Точка завершения третьего действия и начала четвертого отмечена как 'Точка сохранения B'.

Стрелки под многоцветной полоской направлены налево и представляют влияние точек сохранение на откат транзакции:

- Первая стрелка указывает на точку сохранения B. Откат транзакции до точки сохранения B отменяет только последнее действие (синий участок под названием 'Удаление')
- Вторая стрелка указывает на точку сохранения A. Откат транзакции до точки сохранения A отменяет все действия от второго до четвертого (синий участок 'Обновление', желтый участок 'Вставка' и синий участок 'Удаление')
- Последняя стрелка указывает на начало транзакции. Полный откат транзакции отменяет все действия

Запуск операторов SQL с помощью объектов Statement:

Объект Statement позволяет запустить оператор SQL и получить таблицу результатов, если она нужна.

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”: Выполняет простой оператор SQL без параметров.
- “Интерфейс PreparedStatement” на стр. 74 - Выполняет подготовленный оператор SQL с входными параметрами или без них.
- “Интерфейс CallableStatement” на стр. 65 - 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:

- Выполнять различные типы операторов
- Получать значения различных параметров объекта Statement, включая следующие:
  - Соединение
  - Любой из ключей, автоматически созданных во время выполнения Statement
  - Размер выборки и направление выборки
  - Максимальный размер поля имаксимальное количество строк
  - Текущий набор результатов, следующий набор результатов, тип набора результатов, уровень параллелизма набора результатов и тип блокировки курсора набора результатов
- Добавлять операторы SQL в текущий пакет
- Запускать текущий пакет операторов SQL

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();
```

Управление распределенными транзакциями XA JDBC:

Классы управления распределенными транзакциями XA JDBC позволяют применять драйвер JDBC IBM Toolbox for Java в распределенной транзакции. Распределенными называются транзакции, в которых участвуют несколько источников данных.

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  .

Перечисленные ниже объекты позволяют использовать драйвер JDBC IBM Toolbox for Java в распределенных транзакциях XA:

- 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.

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

Классы заданий IBM Toolbox for Java (из пакета классов доступа) позволяют получать и изменять информацию о задании в программе на Java.

Примечание: IBM Toolbox for Java также содержит классы ресурсов, предоставляющие стандартную среду и согласованный программный интерфейс для работы с различными объектами и списками сервера iSeries. Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий вашему приложению. Для работы с заданиями предназначены классы ресурсов RJob, RJobList и RJobLog.

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

Пакет классов доступа содержит следующие классы заданий:

- Job - позволяет получить и изменить информацию о задании iSeries
- JobList - позволяет получить список заданий системы iSeries
- JobLog - предназначен для работы с протоколом задания системы iSeries

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

```
try {

    // Создание объекта AS400.
    AS400 as400 = new AS400("systemName");

    // Создание объекта Job
    Job job = new Job(as400,"QDEV002");

    // Получение информации о задании
    System.out.println("Владелец задания:" + job.getUser());
    System.out.println("Использование CPU:" + job.getCPUUsed();
    System.out.println("Дата запуска задания : " + job.getJobEnterSystemDate());

    // Разрешение занесения изменений в кэш
    job.setCacheChanges(true);

    // Изменения будут сохраняться в кэше.
```

```
      job.setRunPriority(66);
      job.setDateFormat("*YMD");

      // Фиксация изменений. При этом будет изменено значение в системе
      // iSeries.
      job.commitChanges();

      // Передача информации о задании в систему напрямую, минуя кэш.
      job.setCacheChanges(false);
      job.setRunPriority(60);
  } catch (Exception e)
  {
      System.out.println(quot;error :" + e)
  }
```

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. 501

“Пример: Получение списка заданий с помощью объекта JobList” на стр. 503

“Пример: Просмотр сообщений протокола заданий с помощью объекта JobLog” на стр. 507

Класс Job:

Класс Job (в пакете доступа) позволяет программе на Java получать и изменять данные задания сервера.

Примечание: IBM Toolbox for Java также содержит классы ресурсов, предоставляющие стандартную среду и согласованный программный интерфейс для работы с различными объектами и списками сервера iSeries. Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий вашему приложению. Для работы с заданиями предназначены классы ресурсов RJob, RJobList и RJobLog.

В частности, с их помощью можно получить информацию о следующих объектах:
- Очереди задания
- Очереди вывода
- Ведение протоколов сообщений
- Принтер
- Идентификатор страны или региона
- Формат даты

Класс Job также позволяет изменять одно значение за одну операцию и кэшировать несколько изменений с помощью метода setCacheChanges(true) и фиксации изменений с помощью метода commitChanges(). Если кэш не применяется, то фиксировать изменения не нужно.

Пример

Пример программы приведен в справочной документации Java для класса Job. Пример программы позволяет получать и устанавливать значения приоритета запуска в кэше с помощью метода setRunPriority() и задавать формат даты с помощью метода setDateFormat():

Класс Job

Класс JobList:

Класс JobList из пакета классов доступа применяется для создания списков заданий системы iSeries.

Примечание: IBM Toolbox for Java также содержит классы ресурсов, предоставляющие стандартную среду и согласованный программный интерфейс для работы с различными объектами и списками сервера iSeries. Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий вашему приложению. Для работы с заданиями предназначены классы ресурсов RJob, RJobList и RJobLog.

С помощью класса JobList можно получить следующую информацию:

- Полный список заданий
- Список заданий, отсортированный имени, номеру задания или по имени пользователя

С помощью метода getJobs() можно получить список заданий iSeries, а с помощью метода getLength() - число заданий, которые были получены с помощью последнего запущенного метода getJobs().

Пример: Применение метода JobList

Ниже приведен пример получения списка активных заданий системы:

```
                    // Создание объекта AS400. Просмотр заданий
                    // этой системы iSeries.
AS400 sys = new AS400("mySystem.myCompany.com");

                    // Создание объекта, представляющего список заданий.
JobList jobList = new JobList(sys);

                    // Получение списка активных заданий.
Enumeration list = jobList.getJobs();

                    // Печать информации об активных
                    // заданиях.
while (list.hasMoreElements())
{
    Job j = (Job) list.nextElement();

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

Класс JobLog:

Класс JobLog из пакета классов доступа позволяет получать сообщения из протокола задания сервера с помощью метода getMessages().

Примечание: IBM Toolbox for Java также содержит классы ресурсов, предоставляющие стандартную среду и согласованный программный интерфейс для работы с различными объектами и списками сервера iSeries. Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий вашему приложению. Для работы с заданиями предназначены классы ресурсов RJob, RJobList и RJobLog.

Пример: Применение метода JobLog

Ниже приведен пример, в котором печатаются все сообщения из протокола задания текущего пользователя:

```
                    // ... Предполагается, что объект AS/400
                    // и объект списка заданий
                    // уже созданы

                    // Получение списка активных заданий
                    // системы iSeries
Enumeration list = jobList.getJobs();

                    // Выбор задания указанного
```

```
                    // пользователя из списка.
    while (list.hasMoreElements())
    {
       Job j = (Job) list.nextElement();

       if (j.getUser().trim().equalsIgnoreCase(userID))
       {
                    // Задание текущего пользователя
                    // найдено. Создание объекта протокола
                    // для этого задания.
          JobLog jlog = new JobLog(system, j.getName(), j.getUser(), j.getNumber());

                    // Создание списка сообщений из протокола
                    // и печать сообщений.
          Enumeration messageList = jlog.getMessages();

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

       }
    }
```

Классы Message
Класс AS400Message

С помощью объекта AS400Message в программе на Java можно получать сообщения OS/400, созданные при выполнении предыдущей операции (например, при вызове команды). Программа на Java может получить следующую информацию из объекта сообщения:

- Библиотеку iSeries и файл сообщений, которые содержат сообщение
- ИД сообщения.
- Тип сообщения
- Серьезность сообщения
- Текст сообщения
- Справку по сообщению

Ниже приведен пример использования объекта AS400Message:

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

```
                    // 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. 466
- “Пример: Применение объекта ProgramCall” на стр. 519

Класс QueuedMessage

Класс QueuedMessage расширяет класс AS400Message.

Примечание: Toolbox for Java также содержит классы ресурсов, предоставляющие стандартную среду и согласованный программный интерфейс для работы с различными объектами и списками iSeries. Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий вашему приложению. Для работы с сообщениями в очереди предназначен класс ресурсов RQueuedMessage.

С помощью класса QueuedMessage можно получать информацию о сообщениях в очередях сообщений iSeries. Этот класс позволяет получать следующую информацию в программах на Java:

- Информацию об отправителе сообщения, например, имя программы, имя задания, номер задания и имя пользователя
- Очередь сообщений
- Ключ сообщения
- Состояние ответа на сообщение

Ниже приведен пример, в котором печатаются все сообщения из очереди сообщений текущего пользователя:

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

```
                  // Сервер iSeries, на котором находится очередь сообщений.
AS400 sys = new AS400(mySystem.myCompany.com);

                  // Создание объекта очереди сообщений.
                  // Этот объект представляет очередь
                  // текущего пользователя.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                  // Получение списка сообщений из
                  // очереди пользователя.
Enumeration e = queue.getMessages();

                  // Печать всех сообщений из очереди.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

Класс MessageFile

Класс MessageFile позволяет получать сообщения из файла сообщений iSeries. Он возвращает объект AS400Message, содержащий сообщение. С помощью класса MessageFile можно выполнить следующие действия:

- Получить объект, содержащий сообщение
- Получить объект, содержащий текст замещения сообщения

Ниже приведен пример получения и печати сообщения:

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

```
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

Класс MessageQueue позволяет программам на Java работать с очередями сообщений iSeries.

Примечание: Toolbox for Java также содержит классы ресурсов, предоставляющие стандартную среду и согласованный программный интерфейс для работы с различными объектами и списками iSeries. Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий вашему приложению. Для работы с очередями сообщений предназначен класс RMessageQueue.

Класс MessageQueue служит контейнером для класса QueuedMessage. Метод getMessages() возвращает список объектов QueuedMessage. С помощью класса MessageQueue можно выполнить следующие действия:

- Задать атрибуты очереди сообщений
- Получить информацию об очереди сообщений
- Получить сообщения из очереди сообщений
- Отправить сообщения в очередь сообщений
- Ответить на сообщения

Ниже приведен пример получения списка сообщений из очереди текущего пользователя:

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

```
                  // Сервер iSeries, на котором находится очередь сообщений.
AS400 sys = new AS400(mySystem.myCompany.com);

                  // Создание объекта очереди сообщений.
                  // Этот объект представляет очередь
                  // текущего пользователя.
MessageQueue queue = new MessageQueue(sys, MessageQueue.CURRENT);

                  // Получение списка сообщений из
                  // очереди пользователя.
Enumeration e = queue.getMessages();

                  // Печать всех сообщений из очереди.
while (e.hasMoreElements())
{
   QueuedMessage msg = e.getNextElement();
   System.out.println(msg.getText());
}
```

NetServer

Класс NetServer представляет службу NetServer серверов iSeries. Объекты NetServer применяются для просмотра и изменения информации о состоянии и конфигурации службы NetServer.

Например, с помощью класса NetServer можно выполнить следующие операции:

- Запустить или завершить работу NetServer
- Получить список всех текущих общих файлов и общих принтеров

- Получить список всех текущих сеансов работы
- Запросить и изменить значения атрибутов (с помощью методов, унаследованных от класса ChangeableResource)

Примечание: Для применения класса NetServer необходим пользовательский профайл с правами доступа *IOSYSCFG.

Класс NetServer является расширением классов ChangeableResource и Resource, поэтому в нем предусмотрен набор атрибутов, представляющих различные параметры NetServer. Вы можете запросить и изменить значения атрибутов NetServer. Ниже перечислены некоторые из таких атрибутов:

- NAME
- NAME_PENDING
- DOMAIN
- ALLOW_SYSTEM_NAME
- AUTOSTART
- CCSID
- WINS_PRIMARY_ADDRESS

Атрибуты с отложенным изменением

Многие атрибуты NetServer относятся к атрибутам с отложенным изменением (например, 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 - представляет соединение с NetServer
- NetServerFileShare - представляет общий каталог NetServer
- NetServerPrintShare - представляет общий принтер NetServer
- NetServerSession - представляет сеанс NetServer
- NetServerShare - представляет общий ресурс NetServer

Пример: Изменение имени NetServer с помощью объекта NetServer

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

```
// Создание объекта, представляющего сервер iSeries.
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.
```

Классы прав доступа

Классы прав доступа позволяют получать и задавать информацию о правах доступа к объекту. 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

Класс Permission предназначен для получения и изменения информации и правах доступа к объектам. 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(): Добавляет пользователя с правами доступа.
- commit(): Фиксирует изменения прав доступа на сервере.
- getAuthorizationList(): Возвращает список прав доступа для объекта.
- getAuthorizedUsers(): Возвращает перечень пользователей с правами доступа.
- getOwner(): Возвращает имя владельца объекта.
- getSensitivityLevel(): Возвращает уровень защиты данных объекта.
- getType(): Возвращает тип прав доступа к объекту (QDLO, QSYS или Root).
- getUserPermission(): Возвращает права доступа к данному объекту, предоставленные пользователю.
- getUserPermissions(): Возвращает перечень прав доступа к данному объекту, предоставленных пользователям.
- setAuthorizationList(): Задает список прав доступа к объекту.
- setSensitivityLevel(): Задает уровень защиты данных объекта.

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

Класс UserPermission предназначен для задания и изменения прав доступа пользователя к объекту. Класс UserPermission включает три подкласса для работы с разными типами объектов:

Класс UserPermission	Описание
DLOPermission	Задает права доступа пользователя к объектам библиотеки документов (DLO), хранящимся в QDLS.
QSYSPermission	Задает права доступа пользователя к объектам, хранящимся в QSYS.LIB и на сервере.
RootPermission	Задает права доступа пользователя к объектам, находящимся в структуре корневого каталога. К объектам RootPermission относятся те, которые не содержатся ни в QSYS.LIB, ни в QDLS.

Класс UserPermission позволяет выполняет следующие операции:

- Определить, является ли пользовательский профайл профайлом группы
- Получить имя пользовательского профайла.
- Определить, предоставлены ли пользователю права доступа
- Задать права на управление списком прав доступа

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());
}
```

Класс DLOPermission:

Класс DLOPermission является подклассом класса UserPermission.DLOPermission позволяет просматривать и задавать права доступа пользователя к объектам библиотеки документов (DLO).

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.

Права доступа	Описание
*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.

Дополнительная информация приведена в главе 5: Защита ресурсов книги **Справочник по защите iSeries**  .

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());
}
```

Класс QSYSPermission:

QSYSPermission - это подкласс класса UserPermission. Класс QSYSPermission позволяет просматривать и задавать предоставляемые пользователю права доступа к объекту с помощью стандартной библиотечной структуры библиотеки QSYS.LIB системы iSeries. 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.

Системное значение прав доступа	Описание
*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″.

Для получения текущих системных прав доступа предназначен метод getObjectAuthority().Для задания текущих системных прав доступа служит метод setObjectAuthority().

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()

Дополнительная информация о правах доступа приведена в разделе Chapter 5: Resource Security книги **iSeries Security Reference**  .Информация о предоставлении и изменении прав доступа к объектам с помощью команд CL системы iSeries приведена в описании команд Предоставить права доступа к объекту (GRTOBJAUT) и Изменить права доступа к объекту (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());
}
```

Класс RootPermission:

Класс RootPermission является подклассом класса UserPermission.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. Для просмотра текущих значений прав доступа к данным предназначен метод getDataAuthority(), а для задания прав доступа - метод setDataAuthority().

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. Для изменения этих значений служат методы setAlter(), setExistence(), setManagement() и setReference().

После изменения прав доступа к данным или прав доступа к объекту необходимо вызвать метод commit() класса Permissions для сохранения изменений на сервере.

Дополнительная информация о правах доступа приведена в разделе Chapter 5: Resource Security книги **iSeries Security Reference** .

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());
}
```

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. Ресурсы AFP доступны в OS/400 только начиная с версии V3R7. (Попытка открыть список ресурсов AFPResourceList в версии ниже V3R7 приведет к возникновению исключительной ситуации RequestNotSupportedException.)

Классы IBM Toolbox for Java для объектов печати состоят из базового класса PrintObject и шести подклассов для каждого из объектов печати. Базовый класс содержит методы и атрибуты, общие для всех объектов печати сервера. Подклассы содержат методы и атрибуты, относящиеся к отдельному подтипу.

Ниже перечислены операции, которые можно выполнять с помощью классов печати:

- Работа с объектами печати сервера:
  - Класс PrintObjectList служит для получения списков объектов печати сервера и работы с ними. (Объекты печати - это буферные файлы, очереди вывода, принтеры, ресурсы Advanced Function Printing (AFP), файлы принтеров и задания загрузчика.)
  - Базовый класс PrintObject служит для работы с объектами печати.
- Получение атрибутов PrintObject
- Создание буферных файлов сервера с помощью класса SpooledFileOutputStream (для данных принтера в кодировке EBCDIC)
- Генерация потоков данных принтера в виде потока данных SNA (SCS)
- Чтение буферных файлов и ресурсов AFP с помощью PrintObjectInputStream
- Чтение буферных файлов с помощью PrintObjectPageInputStream и PrintObjectTransformedInputStream
- Копирование буферных файлов
- Просмотр буферных файлов Advanced Function Printing (AFP) и потоков символов SNA (SCS)

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

Получение списка объектов печати:

Для работы со списками объектов печати предназначен класс PrintObjectList и его подклассы. Каждый подкласс содержит методы, которые позволяют фильтровать список по критерию, существенному для данного конкретного типа объекта печати. Например, SpooledFileList позволяет отфильтровывать список буферных файлов по имени создавшего их пользователя, по имени очереди вывода, по типу формы или по пользовательским данным буферного файла. В список будут включены только файлы, удовлетворяющие заданным критериям. Если фильтры не установлены, то будут использоваться заданные для них значения по умолчанию.

Для получения из сервера списка объектов печати предназначены методы openSynchronously() и openAsynchronously(). 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. Для этого применяются методы isCompleted() и size(), которые указывают, завершено ли создание списка, и возвращают текущий размер списка. Другие методы, waitForListToComplete() и waitForItem(), позволяют инициатору дождаться окончания создания списка или получения информации о конкретном элементе. 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. Для регистрации или отмены регистрации событий инициатор сначала вызывает метод PrintObjectListListener(), а затем - метод addPrintObjectListListener() для регистрации или метод removePrintObjectListListener() для отмены регистрации. 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.

Событие PrintObjectList	Когда доставляется уведомление о событии
listErrorOccurred	При возникновении любой исключительной ситуации при получении списка.
listOpened	При открытии списка.
listObjectAdded	При добавлении объекта в список.

После открытия списка и обработки его элементов нужно закрыть список с помощью метода close(). 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. Их можно изменить с помощью метода update() из класса PrintObject. Какие именно атрибуты будут пересылаться из сервера, зависит от типа объекта печати, включаемого в список. Для каждого типа объектов печати существует список атрибутов по умолчанию, который можно переопределить с помощью метода setAttributesToRetrieve() из класса PrintObjectList. Списки атрибутов, поддерживаемых для различных типов объектов печати, приведены в разделе Получение атрибутов PrintObject.

Списки ресурсов AFP поддерживаются в OS/400 начиная с версии 3, выпуска 7. Открытие AFPResourceList в системе версии ниже V3R7 приведет к возникновению исключительной ситуации RequestNotSupportedException.

Примеры

Ниже приведены примеры различных способов просмотра списка буферных файлов.

Раздел “Пример: Асинхронное создание списка буферных файлов (с помощью обработчиков событий)” на стр. 513 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. 516 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. 518 содержит информацию о том, как просмотреть список всех буферных файлов системы в синхронном режиме

Работа с объектами печати:

PrintObject - это абстрактный класс. (Абстрактный класс не позволяет создавать экземпляр класса. Вместо этого вы должны создать экземпляр одного из его подклассов.) Создавать объекты подклассов можно следующими способами:

- Если вам известна система и атрибуты объекта, создайте объект с помощью явного вызова общего конструктора.
- Вы можете создать список объектов с помощью подкласса PrintObjectList и работать с отдельными элементами списка.
- Вы можете воспользоваться соответствующим методом, который создает и возвращает объект, или задать вызываемые методы. Например, статический метод start() класса WriterJob возвращает объект WriterJob.

Для работы с объектами печати сервера предназначен базовый класс PrintObject и его подклассы:

- OutputQueue
- Printer

- PrinterFile
- SpooledFile
- WriterJob

Получение атрибутов PrintObject:

Для получения атрибутов объекта печати применяются ИД атрибута и один из следующих методов базового класса PrintObject:

- Метод getIntegerAttribute(int attributeID) - для получения целочисленного атрибута.
- Метод getFloatAttribute(int attributeID) - для получения атрибута с плавающей точкой.
- Метод getStringAttribute(int attributeID) - для получения строкового атрибута.

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. Пользователь может изменить любые атрибуты объекта с помощью метода update(). Кроме того, атрибуты объекта обновляются автоматически, если приложение вызывает методы, приводящие к изменению атрибутов. Например, если атрибут состояния очереди вывода равен RELEASED (метод getStringAttribute(ATTR_OUTQSTS); возвращает строку ″RELEASED″), то при вызове для нее метода hold() атрибут состояния изменится на HELD.

Метод setAttributes

Метод setAttributes применяется для изменения атрибутов буферных файлов и объектов файлов принтеров. Список атрибутов, которые могут задаваться таким образом, приведен в следующих разделах:

- PrinterFileAttrs - атрибуты файлов принтеров
- SpooledFileAttrs - атрибуты буферных файлов

В методе setAttributes предусмотрен параметр PrintParameterList, который задает класс для хранения набора идентификаторов атрибутов и их значений. Сразу после создания список пуст, и вызывающий объект может добавлять в него атрибуты с помощью различных методов setParameter().

Класс PrintParameterList

Класс PrintParameterList служит для передачи группы атрибутов в метод, который далее использует любые из них в качестве параметров. Например, вы можете передать буферный файл с помощью TCP (LPR), используя метод SpooledFile 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.

Атрибуты ресурсов AFP:

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:

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:

Получение атрибутов

Ниже перечислены атрибуты принтера, которые можно получить с помощью методов getIntegerAttribute), getStringAttribute() и getFloatAttribute():

- 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:
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:

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
- scope="local"> ATTR_IPP_ATTR_CHARSET - Атрибуты IPP - набор символов
- scope="local"> ATTR_IPP_JOB_ID - ИД задания IPP
- scope="local"> ATTR_IPP_JOB_NAME - Имя задания IPP
- scope="local"> ATTR_IPP_JOB_NAME_NL - NL имени задания IPP
- scope="local"> ATTR_IPP_JOB_ORIGUSER - Инициатор задания IPP
- scope="local"> ATTR_IPP_JOB_ORIGUSER_NL - NL инициатора задания IPP

- scope="local"> ATTR_IPP_PRINTER_NAME - Имя принтера IPP
- ATTR_JOBNAME - Имя задания
- ATTR_JOBNUMBER - Номер задания
- ATTR_JOBUSER - Пользователь задания
- scope="local">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
- scope="local"> ATTR_RPLUNPRT - Замена непечатаемых символов
- ATTR_RPLCHAR - Символ замещения
- ATTR_RESTART - Повторение печати
- ATTR_SADDLESTITCH_NUMSTAPLES - Число скрепок скрепления посередине
- scope="local"> ATTR_SADDLESTITCH_REF - Базовый край скрепления посередине
- ATTR_SAVE - Сохранение буферного файла
- scope="local"> 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
- scope="local"> ATTR_TIME_WTR_BEGAN_FILE - Время начала сохранения буферного файла

- scope=″local″> ATTR_TIME_WTR_CMPL_FILE - Время завершения сохранения буферного файла
- ATTR_PAGES - Всего страниц
- ATTR_UNITOFMEAS - Единица измерения
- scope=″local″> ATTR_USERCMT - Комментарий пользователя
- ATTR_USERDATA - Пользовательские данные
- ATTR_USRDEFDATA - Данные, определенные пользователем
- ATTR_USRDEFFILE - Файл, определенный пользователем
- ATTR_USRDEFOPT - Пользовательские опции
- scope=″local″> ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS

Указание атрибутов

Ниже перечислены атрибуты буферного файла, которые можно задать с помощью метода setAttributes():
- scope=″local″> 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
- scope=″local″> ATTR_PRTSEQUENCE - Порядок печати
- scope=″local″> 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
- scope=″local″> ATTR_USER_DEFINED_OBJECT - Имя пользовательского объекта в IFS

Атрибуты загрузчика:

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:

Оглавление

- Advanced Function Printing
- Ресурс AFP
- 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:

Создать копию буферного файла, который представляет объект SpooledFile, можно с помощью метода копирования класса SpooledFile. 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 OS/400. 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 обслуживания 

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. Дополнительная информация об API QSPCRTSP приведена в следующем разделе:

API Создать буферный файл (QSPCRTSP)

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(<имя-очереди-вывода>);
```

, где <имя-очереди-вывода> - это объект OutputQueue.

```
public static void main(String args[]) {
   // Создание объекта в системе
   AS400 as400 = new AS400(<имя-системы>,<имя-пользователя>, <пароль>);
   // Указание очереди сообщения, содержащей исходный файл.
   OutputQueue outputQueue =
     new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<имя-очереди-вывода>.OUTQ");

   // Создание массива, содержащего все элементы, необходимые для
   // идентификации уникального буферного файла в системе iSeries.
   String[][] splfTags = { {
      <имя-буферного-файла>,
      <номер-буферного-файла>,
      <имя-задания>,
      <имя-пользователя>,
      <номер-задания>,
      // Значения <имя-системы>,<дата> и <время> не являются обязательными.
      // Если они не указаны, необходимо удалить соответствующие теги
      // splfTags[i],[j], где j может принимать значения 5,6 или 7.
      <имя-системы>,
      <дата>,
      <время>},
   };

   // Сохранение информации, идентифицирующей буферный файл, в файле System.out
   for ( int i=0; i<splfTags.length; i++) {
      System.out.println("Копирование -> " + 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(<имя-очереди-вывода>);
   // где <имя-очереди-вывода> - это объект OutputQueue. Очередь вывода можно
   // задать с помощью следующих команд:
   // OutputQueue outputQueue =
   //    new OutputQueue(as400, "/QSYS.LIB/QUSRSYS.LIB/<имя-очереди-вывода>.OUTQ");
   try { SpooledFile newSplf = new sourceSpooledFile.copy();
   }

   catch (Exception e) {
   }
```

Справочная документация по Java

Дополнительная информация о методе SpooledFile.copy() приведена в следующих разделах справочной документации по Java:

Метод SpooledFile.copy()

Создание буферных файлов:

Для создания буферного файла сервера предназначен класс SpooledFileOutputStream. 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. Полный список атрибутов, которые могут быть заданы в PrintParameterList при создании буферного файла, приведен в документации по конструктору SpooledFileOutputStream.

Для записи данных в буферный файл служат методы write(). 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)
- Она ускоряет обработку потока вывода, поскольку соединение с сервером устанавливается не для каждого запроса на запись.

Для принудительной отправки данных на сервер предназначен метод flush().

После того как инициатор завершает запись данных в новый буферный файл, для закрытия этого файла вызывается метод close(). Если буферный файл закрыт, запись в него невозможна. После того как буферный файл был закрыт, инициатор может получить ссылку на объект SpooledFile, представляющий буферный файл, с помощью метода getSpooledFile().

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 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.

Примеры

Ниже приведены примеры работы с буферными файлами. Первый пример посвящен созданию на сервере буферного файла из потока входных данных. Во втором примере показано, как с помощью класса SCS3812Writer можно создать поток данных SCS и сохранить его в буферном файле на сервере.

“Пример: Создание буферных файлов” на стр. 508

“Пример: Создание буферных файлов SCS” на стр. 509

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 - это поток текстовых данных в кодах EBCDIC, которые могут печататься на принтерах SCS, IPDS или PC.) 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:

Класс загрузчиков SCS	**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).

Класс загрузчиков SCS	Описание
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.

После создания загрузчика можно выполнять вывод текста. Для этого служат методы write(). Для перемещения курсора записи предназначены методы carriageReturn(), lineFeed() и newLine(). Для перехода к новой странице предназначен метод endPage().

После записи всех данных вызовите метод close() для закрытия потока вывода.

Пример

В приведенном ниже примере показано, как с помощью класса SCS3812Writer можно создать поток данных SCS и сохранить его в буферном файле на сервере.

Пример: Создание буферных файлов SCS

Чтение буферных файлов и ресурсов AFP:

Для чтения содержимого буферного файла или ресурса Advanced Function Printing (AFP) с сервера предназначен класс PrintObjectInputStream. Этот класс представляет собой расширение стандартного класса java.io.InputStream JDK, поэтому он может применяться везде, где применяется класс InputStream.

Получить объект PrintObjectInputStream можно с помощью метода getInputStream() экземпляра класса SpooledFile или метода getInputStream() экземпляра класса AFPResource. Просмотр потока ввода буферного файла поддерживается в OS/400 версий V3R2, V3R7 и выше. Просмотр потока вывода ресурсов AFP поддерживается начиная с версии V3R7.

Для чтения данных из потока ввода предназначены методы read(). Все эти методы возвращают либо число считанных байт, либо -1, если не было считано ни одного байта или был достигнут конец файла.

Для получения числа байт в буферном файле или ресурсе AFP предназначен методavailable() из класса PrintObjectInputStream. Класс PrintObjectInputStream поддерживает создание потока ввода, поэтому PrintObjectInputStream всегда возвращает значение true при применении метода markSupported(). Для перемещения текущей позиции чтения назад в потоке ввода могут применяться методы mark() и reset(). Для перемещения позиции чтения в потоке ввода вперед без чтения данных предназначен метод skip().

Пример

Ниже приведен пример чтения данных из существующего буферного файла на сервере с помощью класса PrintObjectInputStream

Пример: Чтение буферных файлов

Чтение данных из буферных файлов с помощью PrintObjectPageInputStream и PrintObjectTransformedInputStream:

Для постраничного чтения данных из буферных файлов AFP и SCS сервера предназначен класс PrintObjectPageInputStream.

Для получения объекта PrintObjectPageInputStream применяется метод getPageInputStream().

Для чтения данных из потока ввода предназначены методы read(). Все эти методы возвращают либо число считанных байт, либо -1, если не было считано ни одного байта или был обнаружен конец страницы.

Для получения размера текущей страницы в байтах предназначен метод available() из класса PrintObjectPageInputStream. Класс PrintObjectInputStream поддерживает создание потока ввода, поэтому PrintObjectInputStream всегда возвращает значение true при применении метода markSupported(). Для перемещения текущей позиции чтения назад в потоке ввода инициатор применяет методы mark() и reset(). Инициатор может переместить позицию чтения в потоке ввода вперед, не считывая данные, с помощью метода skip().

Однако для преобразования всего потока данных буферного файла применяется класс PrintObjectTransformedInputStream.

Пример

Ниже приведен пример преобразований данных, полученных из буферного фала, с помощью классов PrintObjectPageInputStream и PrintObjectTransformedInputStream:

"Пример: Чтение и преобразование буферных файлов" на стр. 512

Лицензия на продукт

Класс ProductLicense позволяет запрашивать лицензии на продукты, установленные в системе iSeries. Для того чтобы избежать конфликтов с другими пользователями лицензии, класс запрашивает и освобождает лицензии с помощью функций работы с лицензиями системы iSeries.

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.

Класс ProductLicense в сочетании с функциями работы с лицензиями iSeries могут применяться для поддержки лицензионных приложений:

- Приложение-сервер регистрирует продукт и лицензионное соглашение с помощью функций работы с лицензиями iSeries.
- 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. Эти сведения сохраняются в системе iSeries с помощью функций работы с лицензиями. 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.

Дополнительная информация и примеры программ приведены в разделе Документация Java для класса ProductLicense.

Класс ProgramCall

Класс ProgramCall предназначен для вызова программ iSeries из программы на Java. С помощью класса ProgramParameter можно задать параметры ввода, вывода и ввода-вывода. При выполнении программы параметры вывода и ввода-вывода будут содержать данные, возвращаемые программой iSeries. В случае сбоя программы iSeries программа на Java может получить сообщения об ошибках iSeries в виде списка объектов AS400Message.

Обязательные параметры:

- Имя выполняемой программы и параметры
- Объект AS400, представляющий сервер iSeries, на котором расположена программа.

Имя программы и список параметров можно задать с помощью метода setProgram() конструктора или метода run(). Этот методы выполняет вызов программы.

При применении класса ProgramCall объект AS400 подключается к системе iSeries. 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.

По умолчанию программы iSeries выполняются в отдельном задании сервера, даже если программа на Java работает на одном сервере с программой iSeries. С помощью метода setThreadSafe() можно указать, что программа iSeries должна выполняться в задании Java.

Применение объектов ProgramParameter

С помощью объектов ProgramParameter можно передавать значения параметров из программ Java в программы iSeries и обратно. Входные параметры задаются с помощью метода setInputData(). Для получения вывода программы после ее выполнения служит метод getOutputData(). Каждый параметр представляет собой массив байтов. Программа на Java должна преобразовывать эти массивы из формата Java в формат iSeries и обратно. Методы преобразования данных содержатся в классах 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.

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

                    // Данная программа использует два параметра.
                    // Для хранения этих параметров создается
                    // список.
ProgramParameter[] parmList = new ProgramParameter[2];

                    // Первый параметр -
                    // входной
byte[] key = {1, 2, 3};
parmList[0] = new ProgramParameter(key);

                    // Второй параметр - выходной.
                    // параметра. Возвращается
                    // четырехбайтовое значение.
parmList[1] = new ProgramParameter(4);

                    // Создание объекта программы.
                    // Задаются имя программы и список
                    // параметров.
ProgramCall pgm = new ProgramCall(sys, "/QSYS.LIB/MYLIB.LIB/MYPROG.PGM", parmList);

                    // Запуск программы.
if (pgm.run() != true)
{

                    // Если в системе iSeries не удалось запустить
                    // программу, просмотрите список сообщений
                    // и найдите причину ошибки.
   AS400Message[] messageList = pgm.getMessageList();

}
else
{
                    // Программа выполнена. Обрабатывается
                    // второй параметр, содержащий
                    // возвращаемые данные.

                    // Объект-преобразователь
```

```
                    // для данного типа данных iSeries
    AS400Bin4 bin4Converter = new AS400Bin4();

                    // Преобразование типа iSeries в объект Java.
                    // Число расположено в начале
                    // буфера.
    byte[] data = parmList[1].getOutputData();
    int i = bin4Converter.toInt(data);
}

                    // Отключение после завершения
                    // программ
sys.disconnectService(AS400.COMMAND);
```

Класс QSYSObjectPathName

Класс QSYSObjectPathName предназначен для определения пути к объектам в интегрированной файловой системе. 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
```

Пример 2: Если имя объекта AS400 применяется только один раз, программа на Java может создать имя с помощью метода toPath(). 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. Для разбиения этого имени на составные части можно использовать класс QSYSObjectPathName:

```
                    // 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();
```

Доступ на уровне записей

Классы доступа на уровне записей позволяют выполнять следующие функции:

- Создавать физический файл iSeries, указывая для этого:
  - Длину записи
  - Исходный файл спецификаций описаний существующих данных (DDS)
  - Объект RecordFormat
- Получать формат записи из физического или логического файла iSeries, либо набор форматов записей из логического файла iSeries с несколькими форматами.

  Примечание: Сведения о формате записей передаются не полностью. Получаемая информация о форматах записей предназначается для настройки формата записи для объекта AS400File. Запрашивается только та информация, которая необходима для описания содержимого записи в файле. Например, информация о заголовках колонок и о псевдонимах для этого не нужна, поэтому она не передается.
- Устанавливать доступ к записям в файле iSeries: последовательно, по номеру записи или по ключу.
- Сохранять записи в файле iSeries.
- Обновлять записи в файле iSeries: последовательно, по номеру записи или по ключу.
- Удалять записи в файле iSeries: последовательно, по номеру записи или по ключу.
- Блокировать доступ к файлу iSeries в соответствии с различными типами прав доступа.
- Устанавливать режим управления фиксацией, позволяющий программе на Java выполнять следующие операции:
  - Включать для соединения опцию управления фиксацией.
  - Устанавливать для разных файлов различные уровни блокировки.
  - Выполнять фиксацию и откат транзакций.
- Удалять файлы iSeries.
- Удалять элементы файлов iSeries.

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.
- Класс KeyedFile реализует доступ к файлу iSeries по ключу.
- Класс SequentialFile реализует доступ к файлу iSeries по номеру записи.
- Класс AS400FileRecordDescription включает методы получения информации о формате записи в файле iSeries.

Классы доступа на уровне записей требуют, чтобы был создан объект AS400, соответствующий той системе, в которой находятся файлы базы данных. Классы доступа на уровне записей устанавливают соединение объекта AS400 с системой iSeries. 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
- Пример последовательного доступа показывает порядок установки последовательного доступа к файлу iSeries.
- Пример чтения файла иллюстрирует способ использования классов доступа на уровне записей для чтения файла iSeries.
- Пример доступа по ключу иллюстрирует использование классов доступа на уровне записей для чтения записей файла iSeries по ключу.

Класс AS400File:

Класс AS400File включает методы для выполнения следующих функций:
- 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

Класс KeyedFile:

Класс KeyedFile позволяет программе на Java обращаться к файлу сервера по ключу. Доступом по ключу называется режим доступа, 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[]) - устанавливает курсор на первую запись с указанным ключом.
- positionCursorAfter(Object[]) - устанавливает курсор на запись, следующую после первой записи с указанным ключом.

- positionCursorBefore(Object[]) - устанавливает курсор на запись, предшествующую первой записи с указанным ключом.

Для удаления записи предназначен следующий метод:
- deleteRecord(Object[]) - удаляет первую запись с указанным ключом.

Для чтения записи предназначены следующие методы:
- read(Object[]) - считывает первую запись с указанным ключом.
- readAfter(Object[]) - считывает запись, следующую после первой записи с указанным ключом.
- readBefore(Object[]) - считывает запись, предшествующую первой записи с указанным ключом.
- readNextEqual() - считывает следующую запись, ключ которой совпадает с указанным ключом. Поиск начинается с записи, которая находится после текущей позиции курсора.
- readPreviousEqual() - считывает предыдущую запись, ключ которой совпадает с указанным ключом. Поиск начинается с записи, которая находится перед текущей позицией курсора.

Для обновления записи предназначен следующий метод:
- update(Object[]) - обновляет запись с указанным ключом.

Класс также содержит методы для указания критерия поиска при установке курсора, чтении или обновлении по ключу. Допустимы следующие значения критерия поиска:
- Равно - поиск первой записи, ключ которой совпадает с указанным ключом.
- Меньше - поиск последней записи, ключ которой расположен перед указанным ключом в последовательности ключей файла.
- Меньше или равно - поиск первой записи, ключ которой совпадает с указанным ключом. Если такие записи отсутствуют, выполняется поиск последней записи, ключ которой расположен перед указанным ключом в последовательности ключей файла.
- Больше - поиск первой записи, ключ которой расположен после указанного ключа в последовательности ключей файла.
- Больше или равно - поиск первой записи, ключ которой совпадает с указанным ключом. 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.

Значения ключевых полей записи могут быть получены из объекта Record файла с помощью метода getKeyFields().

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);
```

Класс SequentialFile:

Класс SequentialFile позволяет программе на Java обращаться к данным файла сервера по номеру записи. Класс содержит методы для установки курсора, чтения, обновления и удаления записи по ее номеру.

Для установки курсора предназначены следующие методы:

- positionCursor(int) - устанавливает курсор на запись с указанным номером.
- positionCursorAfter(int) - устанавливает курсор на запись, следующую после записи с указанным номером.
- positionCursorBefore(int) - устанавливает курсор на запись, предшествующую записи с указанным номером.

Для удаления записи предназначен следующий метод:

- deleteRecord(int) - удаляет запись с указанным номером.

Для чтения записи предназначены следующие методы:

- read(int) - считывает запись с указанным номером.
- readAfter(int) - считывает запись, следующую после записи с указанным номером.
- readBefore(int) - считывает запись, предшествующую записи с указанным номером.

Для обновления записи предназначен следующий метод:

- update(int) - обновляет запись с указанным номером.

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);
```

Класс AS400FileRecordDescription:

Класс AS400FileRecordDescription содержит методы для получения информации о формате записи в файле сервера. В частности, в классе предусмотрены методы для создания исходного кода Java для подклассов RecordFormat и получения объектов RecordFormat, содержащих описание форматов записей пользовательских физических и логических файлов сервера. 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

Метод createRecordFormatSource() создает для подклассов класса RecordFormat исходные файлы Java. Эти файлы можно откомпилировать и использовать в приложении или аплете как входные данные для метода AS400File.setRecordFormat().

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 на Java, 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

Метод retrieveRecordFormat() возвращает массив объектов RecordFormat, представляющих форматы записей существующего файла сервера. 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. Объект RecordFormat можно затем использовать в качестве входного параметра для метода AS400File.setRecordFormat().

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
                    ....

                    // 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 и удаление файлов и элементов:

Для создания физического файла сервера можно указать длину записи, существующий исходный файл спецификаций описания данных (DDS) или объект RecordFormat.

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(). Перед тем как открыть файл, необходимо задать для него формат записи с помощью метода setRecordFormat(). 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:

- Метод delete() удаляет файлы сервера и все их элементы.
- Метод deleteMember() удаляет только один элемент файла.

Метод addPhysicalFileMember() добавляет элементы в файл.

Чтение и сохранение записей:

Для чтения, записи, обновления и удаления записей из файлов сервера служит класс AS400File. Для доступа к записи используется класс Record, который описывается классом RecordFormat. Задать формат записи можно с помощью метода setRecordFormat(). Формат записи необходимо задавать до открытия файла (кроме тех случаев, когда файл был только что создан (без последующего вызова метода close()) одним из методов create(), в которых устанавливается формат записи для объекта).

Для чтения записи из файла предназначены методы read(). Эти методы выполняют следующие операции:

- read() - чтение записи, на которой установлен курсор
- readFirst() - чтение первой записи в файле
- readLast() - чтение последней записи в файле
- readNext() - чтение следующей записи в файле
- readPrevious() - чтение предыдущей записи в файле

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);
```

Для обновления записи, на которую установлен курсор, предназначен метод update().

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);
```

Для добавления записей в конец файла служит метод write(). В файл можно добавлять одну запись или массив записей.

Для удаления записи, на которой установлен курсор, предназначен метод deleteCurrentRecord().

Блокирование файлов:

Программа на Java может блокировать файл, чтобы другие пользователи не могли обращаться к нему, пока программа не завершит работу с ним. Существуют следующие типы блокировки:

- Блокировка на чтение/Исключительная блокировка - Во время чтения файла текущей программой на Java другим программам запрещено обращаться к файлу.

- Блокировка на чтение/Разрешено совместное чтение - Во время чтения записей из файла текущей программой на Java другим программам разрешено их чтение.
- Блокировка на чтение/Разрешена совместная запись - Во время чтения записей из файла текущей программой на Java другим программам разрешено изменять файл.
- Блокировка на запись/Исключительная блокировка - Во время изменения файла текущей программой на Java другим программам запрещено обращаться к файлу.
- Блокировка на запись/Разрешено совместное чтение - Во время изменения файла текущей программой на Java другим программам разрешено чтение записей из файла.
- Блокировка на запись/Разрешена совместная запись - Во время изменения файла текущей программой на Java другим программам разрешено изменять его.

Для освобождения блокировок, полученных с помощью метода lock(), программа на Java вызывает метод releaseExplicitLocks().

Объединение записей в блоки:

Для повышения производительности класс AS400File применяет объединение записей в блоки:

- Если файл открыт только для чтения, то при считывании записи программой на Java считывается блок записей. Применение блоков записей повышает быстродействие программы, так как позволяет обрабатывать запросы на чтение последующих записей без обращения к серверу. 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);
```

- Если файл открыт только для записи, число записей в блоке указывает, сколько записей должно заноситься в файл при каждом вызове метода write(Record[]).

  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.

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:

- Метод positionCursorAfterLast() - устанавливает курсор после последней записи. Если применяется данный метод, то для обращения к записям в файле программа на Java может использовать метод readPrevious().
- Метод positionCursorBeforeFirst() - устанавливает курсор перед первой записью. Если применяется данный метод, то для обращения к записям в файле программа на Java может использовать метод readNext().
- Метод positionCursorToFirst() - устанавливает курсор на первую запись.
- Метод positionCursorToLast() - устанавливает курсор на последнюю запись.
- Метод positionCursorToNext() - устанавливает курсор на следующую запись.
- Метод positionCursorToPrevious() - устанавливает курсор на предыдущую запись.

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);
```

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.

Существуют следующие уровни управления фиксацией:

- All - Пока транзакция не будет зафиксирована или аннулирована, все записи в файле блокируются.
- Change - Пока транзакция не будет зафиксирована или аннулирована, в файле блокируются обновленные, добавленные и удаленные записи.

- Cursor Stability - Пока транзакция не будет зафиксирована или аннулирована, в файле блокируются обновленные, добавленные и удаленные записи. Если приложение читает (но не изменяет) запись, то она остается блокированной только до тех пор, пока не произойдет обращение к другой записи.
- None - Режим управления фиксацией для файла не установлен. Изменения в файле немедленно фиксируются и не могут быть аннулированы.

Для запуска режима управления фиксацией предназначен метод startCommitmentControl(). 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. Уровень управления фиксацией для отдельных файлов задается с помощью метода open(). Задайте значение COMMIT_LOCK_LEVEL_DEFAULT, соответствующее уровню управления фиксацией для метода startCommitmentControl().

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.
                    // На этот файл управление фиксацией не распространяется.
                    // Поскольку управление фиксацией для соединения еще не включено,
                    // 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);
```

Метод commit() фиксирует все транзакции с момента последней границы фиксации для **соединения**. Метод rollback() аннулирует все транзакции с момента последней границы фиксации для **соединения**. Для отмены режима управления фиксацией данного соединения служит метод endCommitmentControl(). 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();
```

Вызов служебных программ

Класс ServiceProgramCall предназначен для вызова служебных программ iSeries. ServiceProgramCall - это подкласс класса ProgramCall, применяемого для вызова программ системы iSeries. Для вызова обычных программ iSeries используйте класс ProgramCall.

Класс ServiceProgramCall позволяет вызвать служебную программу iSeries, передав ей данные через входные параметры и получив вывод этой программы через выходные параметры. При применении класса ServiceProgramCall объект AS400 подключается к системе iSeries. Информация об управлении соединениями 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. Для того чтобы служебная программа была запущена в задании Java, воспользуйтесь методом setThreadSafe(), унаследованным из класса ProgramCall.

Применение класса ServiceProgramCall

Перед началом работы с классом ServiceProgramCall убедитесь, что соблюдены следующие требования.

- Служебная программа должна располагаться в системе iSeries
- Служебной программе передается не более семи параметров
- Служебная программа возвращает пустое или численное значение

Работа с объектами ProgramParameter

Для передачи параметров класс ServiceProgramCall применяет класс ProgramParameter. Для передачи входных данных служебной программе iSeries применяется метод setInputData().

Для задания размера вывода применяется метод setOutputDataLength(). Для получения вывода служебной программы после завершения ее работы применяется метод getOutputData(). Помимо самих данных, в классе ServiceProgramCall требуется задать способ их передачи служебной программе. Для этого применяется метод setParameterType() класса ProgramParameter. Атрибут type указывает, как передается параметр: по значению или по ссылке. В любом случае данные передаются от клиента серверу. Сервер iSeries применяет тип параметра для вызова служебной программы после получения данных.

Все параметры будут представлены в виде массива байт. Поэтому для преобразования данных между форматами iSeries и Java необходимо применять классы преобразования и описания данных.

Классы SystemStatus

Классы SystemStatus позволяют получать информацию о состоянии системы, а также получать и изменять параметры системных пулов. Объект SystemStatus содержит методы, позволяющие просматривать информацию о состоянии системы, в том числе:

- getUsersCurrentSignedOn(): Возвращает текущее число пользователей, работающих в системе
- getUsersTemporarilySignedOff(): Возвращает число отключенных интерактивных заданий
- getDateAndTimeStatusGathered(): Возвращает дату и время сбора информации о состоянии системы
- getJobsInSystem(): Возвращает общее число выполняемых пользовательских и системных заданий
- getBatchJobsRunning(): Возвращает число выполняемых пакетных заданий
- getBatchJobsEnding(): Возвращает число завершаемых пакетных заданий
- getSystemPools(): Возвращает список объектов SystemPool, каждый из которых соответствует одному из системных пулов

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();
```

Класс SystemPool:

Класс SystemPool позволяет получать и изменять информацию о системном пуле с помощью следующих методов:

- Метод getPoolSize() позволяет узнать, а метод setPoolSize() - задать размер пула.
- Метод getPoolName() позволяет узнать, а метод setPoolName() - задать имя пула.
- Метод getReservedSize() возвращает объем памяти пула, зарезервированный системой.
- Метод getDescription() возвращает описание системного пула.
- Метод getMaximumActiveThreads() возвращает максимальное число одновременно выполняемых нитей пула.
- Метод setMaximumFaults() задает максимальное число страничных ошибок в секунду для пула.
- Метод setPriority() задает относительный приоритет данного системного пула по сравнению с другими пулами системы.

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());
```

Системные значения

Классы системных значений позволяют программе на Java просматривать и изменять системные значения и сетевые атрибуты. Можно определить собственную группу системных значений.

Объект SystemValue содержит следующую информацию:

- Имя
- Описание
- Выпуск
- Значение

Класс SystemValue позволяет получить системное значение с помощью метода getValue() и изменить системное значение с помощью метода setValue().

Кроме того, можно получить информацию о группе конкретного системного значения:

- Для того чтобы узнать, к какой системной группе относится системное значение, вызовите метод getGroup().

- Для того чтобы узнать, к какой пользовательской группе относится объект SystemValue (если такая группа есть), воспользуйтесь методами getGroupName() и getGroupDescription().

Когда вы впервые считываете системное значение, оно запрашивается из системы iSeries и записывается в кэш. При всех последующих обращениях к системному значению будет использоваться значение из кэша. Для получения значения из системы iSeries, а не из кэша, вызовите метод clear() для очистки текущего кэша.

Список системных значений

Класс SystemValueList представляет список системных значений заданного сервера iSeries. Этот список разделен на несколько системных групп, позволяющих программам на Java получать доступ сразу к нескольким системным значениям.

Группа системных значений

Класс SystemValueGroup представляет пользовательскую группу системных значений и сетевых атрибутов. 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.

После создания группы в нее можно добавлять системные значения с помощью метода add(). Для удаления системных значений из группы применяется метод remove().

После внесения в группу SystemValueGroup всех необходимых системных значений можно получить фактические объекты SystemValue группы с помощью метода getSystemValues(). 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.

Для обновления всего массива объектов SystemValue служит метод refresh().

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.
//Некоторые системные значения могут относиться к разным системам iSeries.
//Способ создания системных значений (с помощью SystemValueGroup или нет) несущественен.
SystemValueGroup.refresh(sysvals);

//Отключение от систем.
sys.disconnectAllServices();
sys2.disconnectAllServices();
```

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	Регистрирует данные, которыми обмениваются система iSeries и программа на Java. 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.

Категория данных	Описание
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.. Выбрав нужные категории, вы можете с помощью метода setTraceOn включать и выключать для них трассировку. Данные записываются в протокол методом log.

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. Для определения текущего состояния трассировки вы можете воспользоваться методом isTraceOn. 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. Для записи протокола в файл вызовите в приложении на Java метод setFileName(). 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");
```

Пользователи и группы

Классы пользователей и групп позволяют программе на Java получить список пользователей и групп системы iSeries, а также информацию об отдельном пользователе.

Примечание: IBM Toolbox for Java также содержит классы ресурсов, предоставляющие стандартную среду и согласованный программный интерфейс для работы с различными объектами и списками сервера iSeries.

Ознакомившись с информацией о классах в пакете доступа и пакете ресурсов, вы можете выбрать объект, наиболее подходящий вашему приложению. Для работы с пользователями предназначены классы ресурсов RUser и RUserList.

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. При обращении к объекту User укажите имя системы с помощью метода setSystem() и имя пользователя с помощью метода setName(). После этого вы можете получить информацию из системы iSeries с помощью метода loadUserInformation().

Объект UserGroup представляет особый тип пользователей, которым соответствуют групповые профайлы. С помощью метода getMembers() можно получить список пользователей, входящих в заданную группу.

Программа на Java может перебрать все элементы множества по их номерам. Все элементы множества являются объектами User, например:

```
// 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

Объект UserList позволяет получить следующие списки объектов:

- Все пользователи и группы
- Только группы
- Пользователи, входящие в группы
- Пользователи, не входящие в группы

Для работы с объектом UserList необходимо указать объект AS400, представляющий систему, с которой вы планируете работать.

По умолчанию выдается полный список пользователей. Точно указать список пользователей можно, используя вместе методы setUserInfo() и setGroupInfo().

Пример: Просмотр списка пользователей выбранной группы с помощью метода UserList.

Класс UserSpace

Класс UserSpace представляет пользовательское пространство на сервере. Обязательные параметры - имя пользовательского пространства и объект AS400, представляющий сервер, на котором находится это пользовательское пространство. С помощью методов этого класса можно выполнять следующие операции:

- Создавать пользовательские пространства.
- Удалять пользовательские пространства.
- Читать информацию из пользовательских пространств.
- Записывать информацию в пользовательские пространства.
- Получать атрибуты пользовательских пространств. Программа на Java может получить начальное значение, размер и параметр автоматического расширения пользовательского пространства.

- Задавать атрибуты пользовательских пространств. Программа на Java может задать начальное значение, размер и параметр автоматического расширения пользовательского пространства.

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);
```

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.

При записи данных трассировки соединения системы iSeries в потоковый файл эти данные сохраняются в двоичном формате. Классы commtrace позволяют работать с различными компонентами потокового файла.

Примечание: Файлы трассировки соединений могут содержать конфиденциальную информацию, например, незашифрованные пароли. Обращаться к файлу трассировки в системе iSeries могут только пользователи со специальными правами доступа *SERVICE. 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.

Наглядное представление структуры данных файла трассировки соединений с помощью классов commtrace приведено на следующей странице:

| “Модель Commtrace”

| В пакет commtrace входят следующие классы:

| “Классы Format и FormatProperties” на стр. 180: Класс Format позволяет считывать как необработанные, | так и отформатированные данные трассировки соединений. FormatProperties задает свойства объекта | Format, такие, как время запуска и завершения, IP-адреса, порты и т.д.

| **Примечание:** Класс Format также можно запустить в виде отдельной программы.

| “Класс Prolog” на стр. 183: Получает данные начального 256-разрядного раздела данных трассировки | соединений iSeries.

| “Класс Frame” на стр. 184: Получает информацию о фреймах данных трассировки соединений.

| “Класс LanHeader” на стр. 184: Получает содержимое раздела данных, который существует в | единичном экземпляре и расположен в начале фрейма. Этот раздел, как правило, содержит сведения об | аппаратном обеспечении, такие, как общие сведения о фрейме - его номер, длину данных и т.д.

| “Класс IPPacket” на стр. 185: Получает данные пакета. IPPacket - это абстрактный родительский класс | пакетов данных различных типов, поддерживаемых пакетом commtrace.

| “Класс Header” на стр. 185: Получает данные заголовка пакета и связанные данные. Заголовок - это | абстрактный родительский класс различных заголовков пакетов, поддерживаемых пакетом commtrace.

| Большинство остальных классов пакета com.ibm.as400.commtrace предназначены для работы с | определенными типами данных. Дополнительная информация о трассировке соединений и всех классах | commtrace приведена на следующих страницах:

| Справочная документация по Java

| Трассировка соединений

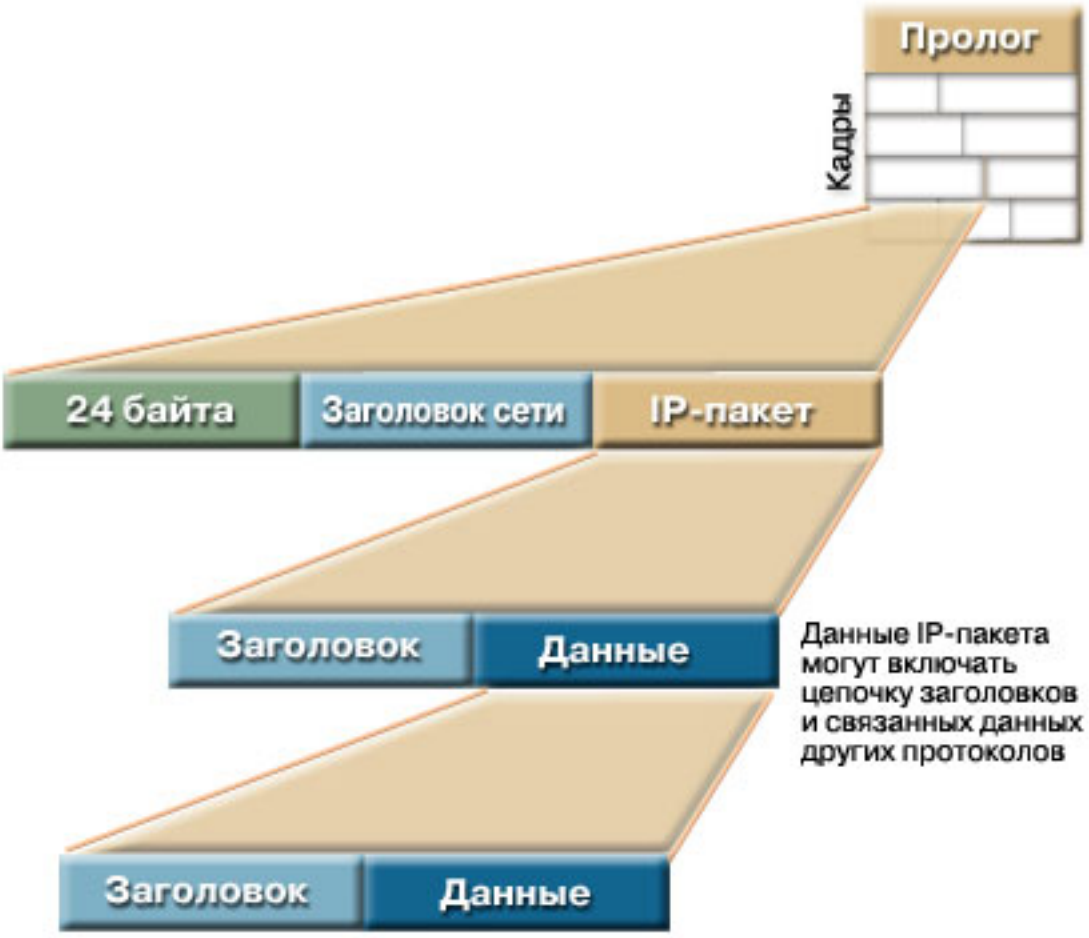
| Модель Commtrace

| 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.

| Prolog: Начальный 256-разрядный раздел трассировки соединений описания линии LAN. Раздел Prolog | содержит общие сведения о трассировке, такие, как время начала и завершения, объем собранных данных и | т. д.

| Frame: Записи переменной длины, которые содержат данные, переданные системой iSeries во время | трассировки соединений.

| **Рисунок 1: Модель Commtrace**



Каждый фрейм файла трассировки состоит из двух начальных разделов (в которых приведена общая информация о содержимом фрейма) и пакета, который система iSeries передала в другой узел сети.

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.

LanHeader: Раздел, который приведен в каждом фрейме только один раз (и расположен после начального 24-разрядного раздела) и содержит общие сведения о следующем разделе IPacket.

IPPacket: Раздел, содержащий один или несколько заголовков и связанные данные. IPPacket представляет пакеты данных, переданных по сети за время трассировки соединений. IPPacket является абстрактным классом, поэтому заголовки и данные пакетов обрабатываются с помощью различных действительных подклассов.

Header: Раздел данных, расположенный в начале IPPacket; он описывает данные следующего за ним пакета и в соответствующих случаях указывает на следующий заголовок. В пакете commtrace Header является абстрактным классом, поэтому для обработки данных применяются различные действительные подклассы.

Подробное описание рисунка 1: Модель Commtrace (rzahh587.gif):

IBM Toolbox for Java: Классы Commtrace

Данный рисунок иллюстрирует общий принцип работы классов commtrace с файлом трассировки соединений.

Описание

Рисунок состоит из следующих компонентов:

- Квадрата на заднем плане в правой части рисунка с меткой 'Prolog'. Этот квадрат разделен на строки, которые представляют раздел трассировки соединений для описания линии локальной сети. Слева от изображения размещена вертикальная надпись 'Frames'.
- Горизонтального прямоугольника на переднем плане в левой части рисунка. Этот прямоугольник соединен с главным квадратом на заднем плане линиями с каждой стороны. Прямоугольник представляет фрейм файла трассировки и состоит из трех разделов:
  - Зеленого раздела с надписью 24 байта, представляющего раздел данных, содержащих общие сведения о содержимом фрейма.
  - Голубого раздела с надписью LanHeader, соответствующего разделу данных, содержащему общие сведения о следующем разделе IPacket.
  - Желтого раздела с надписью IPPacket, который соответствует разделу данных, состоящему из одного или нескольких заголовков и связанных данных.
- Под первым горизонтальным прямоугольником расположен еще один. Он соединен с IPPacket линиями с каждой стороны. Этот прямоугольник содержит подробную схему раздела IPPacket и состоит из двух подразделов:
  - Голубого раздела с надписью Header, соответствующего разделу данных в начале раздела IPPacket.
  - Синего раздела с надписью Data, соответствующего данным пакета и указывающего на следующий заголовок.
- Под синим разделом данных расположен третий горизонтальный прямоугольник. Он соединен с Data линиями с каждой стороны. Прямоугольник представляет два раздела IPPacket и содержит надпись 'IPPacket information encapsulates a chain of other headers and associated data' (Заголовок пакета IPPacket инкапсулирует цепочку других заголовков и связанных данных):
-
  - Голубого раздела с надписью Header, соответствующего разделу данных в начале раздела IPPacket.
  - Синего раздела с надписью Data, соответствующего данным пакета и указывающего на следующий заголовок.

Фрейм (первый горизонтальный прямоугольник) в файле трассировки (фоновое изображение) содержит два раздела данных, 24-разрядный раздел данных (зеленый), раздел LanHeader (голубой), а также пакет (желтый).

IPPacket (второй горизонтальный прямоугольник) во фрейме состоит из разделов Header (голубой) и Data (синий), которые в свою очередь соответствуют другому заголовку и данным (третьему горизонтальному прямоугольнику). Второй и третий прямоугольники соединены линиями с каждой стороны, идущими от раздела Data к IPPacket, что соответствует непрерывной цепочке заголовков и связанных данных при трассировке соединений.

Классы Format и FormatProperties

Класс Format выполняет роль интерфейса между вызывающей программой и фреймами трассировки. Класс FormatProperties позволяет задавать и получать свойства и определять параметры работы объекта Format при получении данных фреймов трассировки.

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(™), 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. 186

Справочная документация по Java

Дополнительная информация о классах Format и FormatProperties приведена в следующих разделах справочной документации по Java:

Format

FormatProperties

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.

Запуск класса format в качестве отдельной программы позволяет использовать при форматировании файлов вычислительные ресурсы и память системы iSeries.

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

Полный список доступных опций приведен в следующем документе:

Справочная документация Java для класса Format

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
}
```

Класс Prolog

Класс Prolog представляет начальный 256-разрядный раздел трассировки соединений описания линии LAN. 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. 186

Справочная документация по Java

Дополнительная информация о классе Prolog приведена в следующем разделе справочной документации по Java:

Класс Prolog

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. Более подробная информация о поддерживаемых форматах данных пакета фрейма и общая информация о структуре фрейма приведена в следующих разделах:

Справочная документация Java для класса Frame

Модель Commtrace

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. С помощью нужного подкласса Header получите данные, связанные с этим заголовком (полезные данные) и все дополнительные заголовки

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. 186

Класс LanHeader

Класс LanHeader представляет раздел данных фрейма, расположенный между начальным 24-разрядным разделом и данными пакета. Этот раздел содержит общую информацию о фрейме.

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

Справочная документация по Java

Дополнительная информация о классе LanHeader приведена в следующем разделе справочной документации Javadoc:

Класс LanHeader

Класс IPPacket

Класс IPPacket - это абстрактный суперкласс, который позволяет создавать классы, представляющие различные типы пакетов. 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

Справочная документация по Java

Дополнительная информация о классах Packet приведена в следующих разделах справочной документации по Java:

Класс IPPacket

Класс ARPPacket

Класс IP4Packet

Класс IP6Packet

Класс UnknownPacket

Класс Header

Класс Header - это абстрактный суперкласс, который позволяет создавать классы, представляющие различные типы заголовков пакетов. 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

Справочная документация по Java

Дополнительная информация о классах Header приведена в следующих разделах справочной документации по Java:

Введение

Класс ARPHeader

Класс ExtHeader

Класс ICMP4Header

Класс ICMP6Header

Класс IP4Header

Класс IP6Header

Класс TCPHeader

Класс UDPHeader

Класс UnknownHeader

Пример: Применение классов Commtrace

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

Классы HTML IBM Toolbox for Java позволяют выполнять следующие действия:

- 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)

Классы HTML реализуют интерфейс HTMLTagElement. Каждый класс создает тег HTML для элемента определенного типа. Тег можно получить с помощью метода getTag(), а затем вставить в любой документ HTML. Теги, создаваемые классами, соответствуют спецификации HTML 3.2.

Классы HTML могут применяться совместно с классами сервлета для получения данных из системы iSeries. 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.

Классы HTML упрощают создание форм, таблиц и других элементов формата HTML:

- Класс BidiOrdering позволяет задавать язык и направление ввода текста.
- Класс DirFilter позволяет узнать, является ли объект типа File каталогом.
- Класс HTMLAlign позволяет выравнивать блоки документа HTML.
- Класс HTMLDocument упрощает создание документов HTML и исходных данных XSL FO.
- Класс HTMLFileFilter позволяет узнать, является ли объект типа File файлом.
- Классы HTMLForm упрощают создание форм (по сравнению со сценариями CGI).
- Класс HTMLHead позволяет создавать теги заголовков страниц HTML.
- Класс HTMLHeading позволяет создавать теги заголовков для страниц HTML.
- Класс HTMLHyperlink упрощает создание ссылок на страницах HTML.
- Класс HTMLImage позволяет создавать теги изображений на страницах HTML.

- Классы HTMLList упрощают создание списков на страницах HTML.
- Класс HTMLMeta позволяет создавать мета-теги для страниц HTML.
- Класс HTMLParameter позволяет указывать параметры HTMLServlet.
- Класс HTMLServlet позволяет создавать расширенные функции сервера.
- Классы HTMLTable упрощают создание таблиц на страницах HTML.
- Класс HTMLText позволяет работать с параметрами шрифтов, применяемых на страницах HTML.
- Классы HTMLTree позволяют вывести иерархический список элементов HTML.
- Класс URLEncoder кодирует разделители для строки URL.
- Класс URLParser позволяет выделять из строки URL значение URI, параметры и описание.

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.

Класс BidiOrdering

Класс BidiOrdering соответствует тегу HTML, изменяющему язык и направление ввода текста. Строка HTML <BDO> имеет два атрибута, один из которых определяет язык, а второй - направление текста.

Класс BidiOrdering позволяет:

- Получать и задавать значение атрибута языка
- Получать и задавать направление ввода текста

Дополнительная информация о применении <BDO> тега HTML приведена на Web-сайте W3C  .

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>Текст на арабском языке.</b>
</bdo>
```

Если страница HTML содержит этот тег, браузеры, поддерживающие тег <BDO>, отображают пример следующим образом:

.txeT cibarA emoS

Класс HTMLAlign

Класс HTMLAlign позволяет выравнивать разделы документа HTML, а не просто его отдельные элементы, например, абзацы и заголовки.

Класс HTMLAlign соответствует тегу <DIV> и его связанным атрибутам выравнивания. Можно задать выравнивание по правому краю, по левому краю или по центру.

С помощью этого класса можно выполнять различные действия, в том числе:

- Добавлять и удалять элементы из списка тегов, которые нужно выровнять
- Получать и задавать тип выравнивания
- Получать и задавать направление интерпретации текста
- Получать и задавать язык элемента ввода
- Получать представление объекта HTMLAlign в виде строки

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>Выровненный по центру неупорядоченный список</li>
  <li>Другой элемент</li>
</ul>
```

На странице HTML этот тег будет выглядеть следующим образом:

- Выровненный по центру неупорядоченный список
- Другой элемент

Класс HTMLDocument

Класс HTMLDocument упрощает создание страниц HTML и документов PDF с помощью существующих классов HTML IBM Toolbox for Java. 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. 14

"Анализатор XML и обработчик XSLT" на стр. 412

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. 192
- "Примеры: Применение класса HTMLDocument" на стр. 194

Справочная документация по Java

Дополнительная информация о классе HTMLDocument приведена в следующем разделе справочной документации по Java:

Класс HTMLDocument

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:

- Создает открывающий тег<HTML>. В конце данных создается закрывающий тег </HTML>.
- Преобразует объекты HTMLHead и HTMLMeta в теги HTML.
- Создает открывающий тег <BODY> сразу после тега <HEAD>. В конце всех данных, непосредственно перед закрывающим тегом </HTML>, создается закрывающий тег </BODY>.

  Примечание: Если вы не указали тег <HEAD>, HTMLDocument создает тег <BODY> сразу после тега <HTML>.
- 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. 197

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

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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:

- Первый раздел содержит теги XSL FO <fo:root> и <fo:layout-master-set>, определяющие общие данные о разметке страницы, такие как высота, ширина и границы. Для того чтобы задать значения параметров разметки, с помощью соответствующих методов HTMLDocument задайте значения связанных свойств.
- Второй раздел содержит тег XSL FO <fo:page-sequence>, в котором размещены отдельные элементы содержимого. 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. 194

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. 197

“Пример: Преобразование исходных данных XSL FO в документ PDF” на стр. 195

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

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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:

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.

Ниже приведен пример одновременного создания исходных данных HTML и XSL FO:

“Пример: Создание источников HTML и XSL FO с помощью класса HTMLDocument.” на стр. 197

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:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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.

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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: Преобразование исходных данных XSL FO в документ PDF:

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.
```

```
//
// 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("Формат:  java ProcessXslFo <имя файла fo> <имя файла pdf>");
            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.:

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

Класс HTMLForm представляет форму HTML. 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. URL действия можно задать с помощью конструктора или метода setURL(). Атрибуты формы могут быть заданы с помощью методов get и получены с помощью различных методов set.

Любой тег HTML можно добавить в объект HTMLForm с помощью метода addElement() и удалить с помощью метода removeElement(). В формах HTMLForms можно применять следующие классы элементов с тегами HTML:

- Классы FormInput: представляют элементы ввода формы HTML
- Классы LayoutFormPanel: позволяют задать способ размещения элементов формы HTML
- TextAreaFormElement: представляет текстовый элемент формы HTML
- LabelFormElement: представляет метку элемента формы HTML
- SelectFormElement: представляет элемент выбора средств ввода формы HTML
- SelectOption: описывает один из элементов списка SelectFormElement
- RadioFormInputGroup: описывает группу радиокнопок, допускающих выбор одного из нескольких вариантов

Кроме того, в форму можно добавить и другие теги, в частности:

- HTMLText
- HTMLHyperlink
- HTMLTable

Для получения дополнительной информации о создании формы с помощью класса HTMLForm ознакомьтесь с данным примером и выводом, полученным после его выполнения.

Классы FormInput:

Класс FormInput позволяет:

- Получить и задать имя элемента ввода
- Получить и задать размер элемента ввода
- Получить и задать начальное значение элемента ввода

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: Представляет переключатель в форме HTML. Программа может изменить или получить текст элемента и его состояние. Переключатель может быть одного из следующих типов:
  - RadioFormInput: Радиокнопка. С помощью класса RadioFormInputGroup можно создать группу радиокнопок, в которой пользователю разрешено выбрать только одну кнопку.
  - CheckboxFormInput: Переключатель, допускающий выбор нескольких элементов в группе. По умолчанию переключатель может быть включен или выключен.

Класс ButtonFormInput:

Класс ButtonFormInput соответствует кнопке в форме HTML.

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=Нажмите здесь onclick="test()" />
```

Класс FileFormInput:

Класс FileFormInput представляет файл в форме HTML.

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" />
```

Класс HiddenFormInput:

Класс HiddenFormInput представляет скрытое поле ввода в форме HTML.

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.

Класс ImageFormInput:

Класс ImageFormInput представляет графический элемент ввода в форме HTML.

Методы класса ImageFormInput позволяют управлять различными атрибутами этого элемента, в частности:

- Получать и задавать исходное изображение
- Получать и задавать способ выравнивания
- Получать и задавать высоту
- Получать и задавать ширину

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" />
```

Класс ResetFormInput:

Класс ResetFormInput соответствует кнопке сброса в форме HTML.

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="Сброс" />
```

Класс SubmitFormInput:

Класс SubmitFormInput представляет кнопку, позволяющую передать информацию, введенную в форме HTML, на обработку.

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="Отправить" />
```

Класс TextFormInput:

Класс TextFormInput представляет однострочное поле ввода текста в форме HTML. Класс TextFormInput содержит методы для получения и задания максимального числа символов в поле ввода.

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="ИД-пользователя" size="40" />
```

Класс PasswordFormInput:

Класс PasswordFormInput предназначен для создания поля ввода пароля в форме HTML.

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" />
```

Класс RadioFormInput:

Класс RadioFormInput соответствует полю ввода радиокнопки в форме HTML. При создании радиокнопку можно инициализировать как выбранную.

Несколько радиокнопок с одинаковым именем управляющего элемента образуют группу. Класс RadioFormInputGroup создает группы радиокнопок. 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" />
```

Класс CheckboxFormInput:

Класс CheckboxFormInput соответствует переключателю в форме HTML. 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:

Класс LayoutFormPanel представляет схему расположения элементов формы HTML. Вы можете воспользоваться методами LayoutFormPanel для добавления и удаления элементов панели и получения общего числа элементов формы. Существует два варианта размещения элементов:

- “GridLayoutFormPanel”: Представляет макет сетки элементов формы HTML
- “Класс LineLayoutFormPanel”: Представляет макет разметки элементов формы HTML

GridLayoutFormPanel:

Класс GridLayoutFormPanel представляет табличную разметку элементов формы. 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>Система:</td>
<td><input type="text" name="Система" /></td>
</tr>
<tr>
<td>Пользователь:</td>
<td><input type="text" name="Пользователь" /></td>
</tr>
<tr>
<td>Пароль:</td>
<td><input type="password" name="Пароль" /></td>
</tr>
</table>
<input type="submit" name="logon" value="Войти в систему" />
</form>
```

Класс LineLayoutFormPanel:

Класс LineLayoutFormPanel представляет линейную схему расположения элементов формы HTML. 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/>
```

Класс TextAreaFormElement:

Класс TextAreaFormElement представляет область ввода текста в форме HTML. Размер области задается с помощью числа строк и столбцов. Узнать текущий размер области можно с помощью методов getRows() и getColumns().

Задать начальный текст можно с помощью метода setText(). Для просмотра заданного начального текста служит метод getText().

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>
```

Класс LabelFormElement:

Класс LabelFormElement соответствует метке элемента формы HTML. Он позволяет создать метку для таких элементов формы HTML, как область текста или поле для ввода пароля. Метка представляет собой строку текста, которая задается с помощью метода setLabel(). 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
```

Класс SelectFormElement:

Класс SelectFormElement представляет поле со списком в форме HTML. Вы можете добавлять и удалять различные элементы этого списка.

В классе SelectFormElement предусмотрены методы для просмотра и изменения атрибутов поля со списком:

- Метод setMultiple() позволяет указать, может ли пользователь выбрать несколько элементов списка
- Метод getOptionCount() позволяет узнать число элементов в списке
- Метод setSize() позволяет задать число выводимых элементов списка, а метод getSize() позволяет определить это число.

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">Вариант1</option>
<option value="opt2">Вариант2</option>
<option value="opt3" selected="selected">Вариант3</option>
</select>
```

Класс SelectOption:

Класс SelectOption представляет опцию класса HTML SelectFormElement. Опции входят в состав поля со списком.

В этом классе предусмотрены методы для получения и изменения атрибутов SelectOption. Например, можно указать, должна ли опция быть выбрана по умолчанию. Вы также можете задать значение опции, которое будет передано при обработке формы.

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">Вариант1</option>
<option value="opt2">Вариант2</option>
<option value="opt3" selected="selected">Вариант3</option>
</select>
```

Класс RadioFormInputGroup:

Класс RadioFormInputGroup представляет группу объектов RadioFormInput. 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:

- Добавлять радиокнопку

- Удалять радиокнопку
- Получать и задавать имя группы радиокнопок

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
```

Класс HTMLHead

Класс HTMLHead представляет тег заголовка HTML. 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. Дополнительная информация о классе HTMLMeta приведена на следующей странице:

Класс 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>Моя главная страница</title>
</head>
```

Справочная документация по Java

Дополнительная информация о работе с классом HTMLHead приведена в следующих разделах справочной документации по Java:

Класс HTMLHead

Класс HTMLHeading

Класс HTMLHeading представляет заголовок HTML. Для заголовка можно задать атрибут выравнивания и уровень от 1 (максимальный шрифт, наибольшая важность) до 6.

Класс HTMLHeading содержит методы, позволяющие:

- Получать и задавать текст заголовка
- Получать и задавать уровень заголовка
- Получать и задавать тег выравнивания заголовка
- Получать и задавать направление чтения текста
- Получать и задавать язык элемента ввода
- Получать представление строки объекта HTMLHeading

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">Заголовок</h1>
<h2 align="center">Подзаголовок</h2>
<h3 align="right">Элемент</h3>
```

Класс HTMLHyperlink

Класс HTMLHyperlink представляет тег гиперссылки HTML. С помощью этого класса вы можете добавить ссылку на страницу HTML. Кроме того, этот класс позволяет получать и задавать следующие атрибуты ссылки:

- Получать и задавать URI ссылки
- Получать и задавать название ссылки
- Получать и задавать целевой фрейм ссылки

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">Домашняя страница IBM Toolbox for Java</a>
```

Класс HTMLImage

Класс HTMLImage позволяет создавать на странице HTML теги изображений. С помощью методов класса HTMLImage можно получать и задавать различные атрибуты изображения, в частности:

- Получать и задавать высоту изображения
- Получить и задавать ширину изображения
- Получить и задавать имя изображения
- Получать и задавать текст, замещающий изображение
- Получать и задавать отступ от границ изображения по горизонтали
- Получать и задавать отступ от границ изображения по вертикали
- Получать и задавать абсолютные и относительные ссылки на изображение
- Получать представление объекта HTMLImage в виде строки

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" />
```

Классы HTMLList

Классы HTMLList упрощают создание списков на страницах HTML. Эти классы позволяют задавать атрибуты списков и их элементов.

Родительский класс HTMLList содержит метод для создания сжатого списка, размер которого по вертикали минимален.

- Класс HTMLList содержит следующие методы:
  - Сжатие списка
  - Добавление и удаление элементов из списка
  - Добавление и удаление списков из списка (только для вложенных списков)
- Класс HTMLListItem содержит следующие методы:
  - Получение и задание содержимого элемента
  - Получение и задание направления интерпретации текста
  - Получение и задание языка элемента ввода

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:
  - Получение и задание номера первого элемента списка
  - Получение и задание типа (стиля) номеров элементов
- Класс OrderedListItem содержит следующие методы:
  - Получение и задание номера элемента
  - Получение и задание типа (стиля) элемента с указанным номером

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:
  - Получение и задание типа (стиля) элементов
- Класс UnorderedListItem содержит следующие методы:
  - Получение и задание типа (стиля) элемента

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>Первый элемент</li>
<li>Второй элемент</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>Первый элемент</li>
<li>Второй элемент</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>Первый элемент</li>
<li>Второй элемент</li>
<ul type="square">
<li>Первый элемент</li>
<li>Второй элемент</li>
</ul>
<li>Третий элемент</li>
</ol>
```

Класс HTMLMeta

Класс HTMLMeta представляет мета-информацию, применяемую в теге HTMLHead. 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:

- Получение и задание атрибута NAME
- Получение и задание атрибута CONTENT
- Получение и задание атрибута HTTP-EQUIV
- Получение и задание атрибута LANG
- Получение и задание атрибута URL

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">
```

Класс HTMLParameter

Класс HTMLParameter представляет параметры, применяемые с классом HTMLServlet. У каждого параметра есть имя и значение.

Методы класса HTMLParameter позволяют:

- Получать и задавать имена параметров
- Получать и задавать значения параметров

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">
```

Класс HTMLServlet

Класс HTMLServlet представляет расширенные функции сервера. В объекте сервлета задается имя сервлета и, при необходимости, его расположение. По умолчанию применяется расположение в локальной системе.

Класс HTMLServlet применяется совместно с классом HTMLParameter, задающим параметры сервлета.

Класс HTMLServlet содержит следующие методы:

- Добавление и удаление объектов HTMLParameter из тега сервлета
- Получение и задание расположения сервлета
- Получение и задание имени сервлета
- Получение и задание альтернативного текста сервлета

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>
```

Классы таблиц HTML

Класс HTMLTable позволяет легко создавать таблицы на страницах HTML. С его помощью можно работать со следующими атрибутами таблицы:

- Получать и задавать ширину рамки
- Получать число строк в таблице
- Добавлять столбец или строку в конец таблицы
- Удалять указанный столбец или строку

Класс HTMLTable при создании таблиц взаимодействует с другими классами. Список этих классов приведен ниже:

- HTMLTableCell: описывает ячейку таблицы

- HTMLTableRow: описывает строку таблицы
- HTMLTableHeader: описывает заголовок-ячейку таблицы
- HTMLTableCaption: описывает название таблицы

Пример: Применение классов HTMLTable

Ниже приведен пример применения классов HTMLTable:

“Пример: Применение классов HTMLTable” на стр. 606

Класс HTMLTableCell:

Класс HTMLTableCell получает на входе любой объект HTMLTagElement и создает тег ячейки таблицы с указанным элементом. Элемент может задать в конструкторе или с помощью одного из двух методов setElement().

Многие атрибуты ячейки можно восстановить или обновить с помощью методов класса HTMLTableCell. С помощью этих методов можно выполнять, например, следующие действия:

- Получать и задавать число строк
- Получать и задавать высоту ячейки
- Указывать, будут ли при работе с данными ячеек применяться стандартные соглашения о разрывах строк HTML

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());
```

Метод getTag() выдаст следующий код:

<td align=″center″>Домашняя страница IBM</td>

Класс HTMLTableRow:

Класс HTMLTableRow создает строку таблицы. Этот класс включает различные методы считывания и задания атрибутов строки. С помощью этих методов можно выполнять, например, следующие действия:

- Добавлять и удалять поля из строки
- Получать данные столбца с указанным индексом
- Получать индекс столбца с указанной ячейкой.
- Получать число полей в строке
- Задавать выравнивание по горизонтали и вертикали

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);
```

Класс HTMLTableHeader:

Класс HTMLTableHeader унаследован от класса HTMLTableCell. Он создает ячейку особого типа, ячейку-заголовок, задаваемую тегом **<th>** вместо **<td>**. 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);
```

Класс HTMLTableCaption:

Класс HTMLTableCaption служит для создания названий таблиц HTML. Этот класс содержит методы для обновления и восстановления атрибутов названия. Метод setAlignment() позволяет задавать выравнивание для названия таблицы. 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);
```

Класс текста HTML

Класс HTMLText позволяет работать с различными свойствами текста, размещенного на странице HTML. С помощью класса HTMLText программист может управлять атрибутами, в частности:

- Получать и задавать размер шрифта
- Включать и выключать атрибут полужирного текста и определять его текущее состояние
- Включать и выключать атрибут подчеркивания и определять его текущее состояние
- Получать и задавать атрибут горизонтального выравнивания текста

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

Классы HTMLTree

Класс HTMLTree позволяет легко создавать иерархические деревья элементов, применяемые на страницах HTML. Помимо методов для работы с атрибутами дерева, этот класс содержит методы для:

- Получения и добавления запроса сервлета HTTP
- Добавления элементов HTMLTreeElement или FileTreeElement в дерево
- Удаления элементов HTMLTreeElement или FileTreeElement из дерева

Для упрощения создания иерархических деревьев класс HTMLTree применяется совместно с другими классами HTML:

- HTMLTreeElement: Применяется для создания элемента дерева
- FileTreeElement: Применяется для создания элемента дерева файлов
- FileListElement: Применяется для создания элемента списка файлов
- FileListRenderer: Применяется для вывода списка файлов и каталогов

Примеры: Применение классов HTMLTree

Различные способы применения классов HTMLTree продемонстрированы в следующих примерах:

- “Пример: Применение классов HTMLTree” на стр. 596
- “Пример: Создание просматриваемого дерева интегрированной файловой системы”

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:

Класс HTMLTreeElement представляет элемент иерархии HTMLTree или другого объекта HTMLTreeElement.

Многие атрибуты этого дерева можно получить или изменить с помощью методов класса HTMLTreeElement. С помощью этих методов можно выполнять, например, следующие действия:

- Получать и задавать текст элемента
- Получать и задавать URL развернутого и свернутого значка
- Указывать, следует ли расширять элемент дерева

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());
```

В примере, приведенном выше, с помощью метода getTag() генерируются следующие теги HTML:

```
<table cellpadding="0" cellspacing="3">
<tr>
<td><font color="#0000FF"><u>-</u></font> </td>
<td><font color="#0000FF"><u>Моя Web-страница</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>Другая Web-страница</u></font> </td>
</tr>
</table>
</td>
</tr>
</table>
```

Класс FileTreeElement:

Класс FileTreeElement применяется для представления Интегрированной файловой системы (IFS) в виде HTMLTree.

Многие атрибуты этого дерева можно получить или изменить с помощью методов HTMLTreeElement. Вы также можете получить и задать имя и путь к общим дискам NetServer.

Ниже перечислены некоторые действия, которые можно выполнить с помощью этих методов:

- Получить и задать адрес развернутого и свернутого значка (наследуемый метод)
- Установите, следует ли расширять элемент дерева (унаследованный метод)
- Получать или задавать имя общего диска NetServer
- Получать или задавать путь к общему диску NetServer

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());
```

Приведенный выше метод getTag() создает вывод данного примера.

Класс FileListElement:

Класс FileListElement позволяет создать список объектов каталога интегрированной файловой системы.

С помощью объекта FileListElement можно представить содержимое общего диска NetServer, получив и задав имя и путь к общим дискам NetServer.

Класс FileListElement содержит методы, позволяющие:

- Показывать и сортировать элементы списка файлов
- Получать и задавать запрос сервлета HTTP
- Получать и задавать FileListRenderer
- Получать и задавать объект HTMLTable, с помощью которого отображается список файлов
- Получать или задавать имя общего диска NetServer
- Получать или задавать путь к общему диску NetServer

Класс FileListElement можно применять совместно с другими классами из пакета HTML:

- С помощью класса FileListRenderer можно указать способ отображения списка файлов
- С помощью класса FileTreeElement можно создать просматриваемый список интегрированных файловых систем или общих файлов NetServer

Информация о создании объекта FileListElement и просмотре его содержимого приведена в разделе документации по языку Java, посвященному классу FileListElement.

Пример: Создание просматриваемого дерева интегрированной файловой системы с помощью класса FileListElement

В приведенном ниже примере показано, каким образом можно создать просматриваемое дерево объектов интегрированной файловой системы с помощью класса FileListElement и классов HTMLTree (FileTreeElement и HTMLTreeElement). Пример также содержит программу, позволяющую задавать путь к общему дереву NetServer.

“Пример: Создание просматриваемого дерева интегрированной файловой системы” на стр. 216

Класс FileListRenderer:

Класс FileListRenderer преобразует любое поле объекта File (каталога или файла) в объект FileListElement.

Методы класса FileListRenderer позволяют выполнять следующие действия:

- Получать имя каталога
- Получать имя файла

- Получать имя родительского каталога
- Получать данные строк для отображения в объекте 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.

Полностью настроить таблицу FileListElement можно с помощью метода getRowData(). 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.

Классы ReportWriter

Пакет com.ibm.as400.util.reportwriter содержит классы, позволяющие при помощи сервера iSeries получать и форматировать данные из исходных файлов XML и данные, созданные сервлетами и JavaServer Pages(TM). 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

“Анализатор XML и обработчик XSLT” на стр. 412

.

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.
- Класс PDFContext совместно с классом ReportWriter применяется для создания отчета в формате PDF.

Классы ReportProcessor из пакета processor предназначены для создания форматированных отчетов на основе информации, собранной приложением в исходных данных XML, сервлетах Java и JavaServer Pages (JSP).

- Класс JSPReportProcessor предназначен для получения данных от сервлетов и страниц JSP и создания отчетов в доступных форматах (контекстах).
- Класс XSLReportProcessor предназначен для обработки данных XSL с помощью стилей XSL и создания отчетов в доступных форматах (контекстах).

Классы Context

Классы Context поддерживают определенные форматы данных, которые совместно с классами OutputQueue и SpooledFileOutputStream используются классами ReportWriter для создания отчетов в этих форматах и размещения этих отчетов в буферном файле.

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). Ниже приведены примеры создания классов Context и их применения совместно с другими классами ReportWriter для создания отчетов:

Пример: Применение класса XSLReportProcessor с PCLContext

Пример: применение класса JSPReportProcessor с PDFContext

Класс JSPReportProcessor

Класс JSPReportProcessor позволяет создавать документы и отчеты на основе JavaServer Page(TM) и сервлетов Java.

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:

- Обрабатывать отчеты
- Задавать URL в качестве шаблона

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. Вы можете также загрузить архивный файл ZIP, который содержит примеры применения классов JSPReportProcessor и XSLReportProcessor для JSP, XML и XSL:

- “Пример: Работа с классом JSPReportProcessor с помощью PDFContext” на стр. 618
- “Пример: Файл JSP для класса JSPReportProcessor” на стр. 619

Дополнительная информация о JSP приведена на Web-сайте Java Server Pages technology .

Класс XSLReportProcessor

Класс XSLReportProcessor предназначен для создания документов и отчетов из файлов в формате XML. 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:

- Создать форму XSL
- Создать источник данных XML
- Создать исходный объект FO XSL
- Обработать отчет

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

Дополнительные сведения об XML и XSL приведены в разделе XML продукта Information Center.

Классы ресурсов

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:

- Resource - объект, соответствующий ресурсу iSeries, например пользователю, принтеру, заданию, сообщению или файлу. Действительные подклассы resource:
  - RIFSFile
  - RJavaProgram
  - RJob
  - RPrinter
  - RQueuedMessage
  - RSoftwareResource
  - RUser

  Примечание: Классы NetServer в пакете доступа также являются действительными подклассами класса Resource.
- ResourceList - объект, соответствующий списку ресурсов iSeries, например, списку пользователей, принтеров, заданий, сообщений или файлов. Действительные подклассы resource:
  - RIFSFileList
  - RJobList
  - RJobLog
  - RMessageQueue
  - RPrinterList
  - RUserList
- Presentation - объект, отвечающий за представление информации об объектах ресурсов, списках ресурсов, атрибутах, выбранных значениях и отсортированных списках конечным пользователям

Классы Resource и ChangeableResource

Абстрактные классы com.ibm.as400.resource.Resource исom.ibm.as400.resource.ChangeableResource соответствуют ресурсам iSeries.

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.

Продукт IBM Toolbox for Java содержит следующие объекты ресурсов:
- Объект RIFSFile соответствует файлу или каталогу в интегрированной файловой системе iSeries.
- Объект RJavaProgram соответствует программе на Java в системе iSeries
- Объект RJob соответствует заданию сервера iSeries
- Объект RPrinter соответствует принтеру iSeries
- Объект RQueuedMessage соответствует сообщению в очереди сообщений или протоколе задания iSeries
- Объект RSoftwareResource соответствует лицензионной программе в системе iSeries
- Объект RUser соответствует пользователю iSeries

Класс ChangeableResource

Абстрактный класс ChangeableResource, дочерний по отношению к классу Resource, позволяет изменять значения атрибутов ресурсов системы iSeries. Измененные значения атрибутов заносятся во внутренний кэш и хранятся в нем, пока не будут зафиксированы или отменены. Это позволяет изменять значения нескольких атрибутов одновременно.

Примечание: Классы NetServer в пакете доступа также являются действительными подклассами классов 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

Отказ от гарантий на предоставляемый код

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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.

Списки ресурсов

Класс com.ibm.as400.resource.ResourceList соответствует списку ресурсов системы iSeries. Этот абстрактный класс обеспечивает общий доступ к информации списка.

Продукт IBM Toolbox for Java содержит следующие объекты списков ресурсов:
- RIFSFileList соответствует списку файлов и каталогов в интегрированной файловой системе iSeries

- RJobList соответствует списку заданий iSeries
- RJobLog соответствует списку сообщений в протоколе задания iSeries
- RMessageQueue соответствует списку сообщений в очереди сообщений iSeries
- RPrinterList соответствует списку принтеров iSeries
- RUserList соответствует списку пользователей iSeries

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:
- Открывать список
- Закрывать список
- Получить доступ к конкретному ресурсу списка
- Дождаться загрузки конкретного ресурса
- Дождаться завершения загрузки списка ресурсов

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:

> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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

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:

- Java Secure Socket Extension (JSSE), обеспечивающая защиту соединений как с помощью шифрования данных сеанса клиент-сервер, так и с помощью идентификации на сервере.

  Примечание: Сведения о применении Secure Sockets Layer (SSL) приведены только для обеспечения совместимости с предыдущими версиями.
- Службы идентификации предназначены для выполнения следующих задач:
  - Идентификации пользователя путем сравнения его имени и пароля со значениями, заданными при регистрации в OS/400.
  - Замены владельца текущей нити OS/400.

SSL

SSL обеспечивает защиту соединений за счет:

- Шифрования данных, передаваемых в сеансе между клиентом и сервером
- Идентификации клиента на сервере

Примечание: Создавать защищенные соединения можно как описанными ниже способами, так и с помощью Java Secure Socket Extension (JSSE). Описание работы SSL приводится только для совместимости со старыми версиями.

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.

Перед началом применения SSL с IBM Toolbox for Java ознакомьтесь со своими правовыми ограничениями.

Алгоритмы SSL

IBM Toolbox for Java не содержит алгоритмов шифрования и расшифровки данных. В операционной системе версии V5R3 эти алгоритмы поставляются в составе лицензионной программы iSeries Client Encryption (128-разрядной), 5722-CE3.

Примечание: Продукт IBM Toolbox for Java также совместим с лицензионной программой iSeries Client Encryption (56-разрядной) 5722-CE2, которая больше не обновляется и в версии V5R3 не поддерживается. Поскольку алгоритмы 128-разрядной версии программы Client Encryption обеспечивают более надежное шифрование, чем 56-разрядной, рекомендуется пользоваться 128-разрядной версией программы Client Encryption.

Информацию о том, как заказать программу Client Encryption (128-разрядную), 5722-CE3, можно получить в представительстве фирмы IBM.

Настройка среды SSL

IBM Toolbox for Java содержит две среды для работы с SSL, требующих настройки.

- Шифрование данных, передаваемых между классами IBM Toolbox for Java и серверами OS/400
- Шифрование данных, передаваемых между клиентом и сервером Proxy

Совместимость с предыдущими версиями IBM Toolbox for Java

Для работы с IBM Toolbox for Java версии V5R3, алгоритмами шифрования и файлами классов наборов ключей необходима лицензионная программа Client Encryption выпуска V5R1, V5R2 или V5R3.

Примечание: При модернизации OS/400 выпуска V4R5 или ниже необходимо обновить файл KeyRing.class.

С помощью IBM Toolbox for Java версии V5R3 и совместимой версии программы Client Encryption можно устанавливать соединения между клиентами и операционными системами OS/400 версии V5R1 и выше. Дополнительная информация о совместимых версиях Client Encryption приведена в разделе Алгоритмы SSL.

Правовые ограничения при работе с SSL: Лицензионный продукт IBM iSeries Client Encryption (128-разрядный) поддерживает 128-разрядный алгоритм шифрования SSL версии 3.0.

Эта программа содержит технологии шифрования данных, на которые распространяются особые правила экспорта Министерства торговли США. В других странах и областях также могут существовать ограничения на экспорт и импорт таких программ.

Обратите внимание, что использование данной программы и передача ее другим пользователям в той же или иной стране или области может быть запрещена или ограничена:

- Особыми законами, правилами или ограничениями на импорт в страну пользователя
- Особыми законами, правилами или ограничениями на экспорт из страны пользователя

Начиная с текущего момента вы полностью принимаете на себя ответственность за использование и распространение программы в соответствии с упомянутыми законами и постановлениями об импорте и экспорте. Эта ответственность не ограничена сроком лицензии на данный продукт.

Все пользователи этой программы должны соблюдать законы об импорте и экспорте, принятые в других странах.

Применение SSL для шифрования данных, передаваемых между IBM Toolbox for Java и серверами OS/400:

Для шифрования данных, передаваемых между классами Java и серверами OS/400, может применяться SSL. В системе клиента для шифрования данных применяются файлы, поставляемые с лицензионной программой IBM iSeries Client Encryption (5722-CE2 или 5722-CE3). Для настройки функции шифрования данных на серверах OS/400 применяется Диспетчер цифровых сертификатов OS/400.

Настройка применения SSL на сервере и на клиенте

Для шифрования данных, передаваемых между классами IBM Toolbox for Java и серверами OS/400, выполните следующие действия:

1. Настройте серверы для работы с зашифрованными данными.
2. Настройте клиент (классы IBM Toolbox for Java) на обмен зашифрованными данными. Для этого выполните процедуру, соответствующую типу сертификата сервера:
   - Работа с сертификатом сервера, выданным уполномоченной сертификатной компанией
   - Применение собственного сертификата

   Примечание: Настраивать клиент с помощью сертификата уполномоченной сертификатной компании значительно проще и быстрее, чем с помощью собственного сертификата.
3. 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. 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.

Настройка серверов iSeries для работы с SSL:

Для того чтобы классы IBM Toolbox for Java могли работать с SSL на сервере iSeries, выполните следующие действия:

1. Установите в системах iSeries следующие продукты:
   - IBM Cryptographic Access Provider (128-разрядная версия) for iSeries, 5722-AC3. Этот продукт обеспечивает поддержку шифрования для серверных приложений.
   - iSeries Client Encryption (128-разрядная версия), 5722-CE3. Этот продукт предоставляет утилиты и классы Java для клиентских приложений IBM Toolbox for Java.

   Примечание: Продукт IBM Toolbox for Java также совместим с лицензионными программами Cryptographic Access Provider iSeries (56-разрядная версия) выпуска V5R1, 5722-AC2, и Client Encryption (56-разрядная версия) выпуска V5R1, 5722-CE2.
2. Измените права доступа к каталогу, содержащему файлы шифрования клиента.
3. Получите и настройте сертификат сервера.
4. Установите сертификат на следующих серверах iSeries, с которыми работает 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

Изменение прав доступа к каталогу, содержащему файлы шифрования клиентских систем

Для соответствия правовым ограничениями при работе с SSL каталог с файлами шифрования клиента поставляется с общими правами доступа *EXCLUDE. Эти права доступа нужно изменить, предоставив доступ только тем пользователям, которые применяют функцию шифрования.

Установите права доступа к файлам шифрования клиента на уровне объектов OS/400, выполнив следующие действия:

1. В системе сервера введите следующую команду:
   ```
   wrklnk '/QIBM/ProdData/HTTP/Public/jt400/*'
   ```
2. Выберите опцию 9 в каталоге SSL56 или SSL128.
3. Убедитесь в том, что права доступа категории *PUBLIC равны *EXCLUDE.
4. Предоставьте права доступа *RX к каталогу пользователям или группам пользователей, которым необходим доступ к файлам SSL.

   Примечание: Запретить доступ к файлам SSL пользователям со специальными правами доступа *ALLOBJ нельзя.

Получение и настройка сертификатов сервера

Перед получением и установкой сертификата сервера необходимо установить следующие продукты:

- Лицензионную программу IBM HTTP Server for iSeries  (5722-DG1)
- 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.), установите его в системе iSeries, а затем примените ко всем серверам хоста.
- Если вы не планируете получать сертификат у уполномоченной сертификатной компании, создайте собственный сертификат iSeries. Создайте сертификат с помощью Диспетчера цифровых сертификатов:
  1. Создайте сертификатную компанию в системе iSeries. Дополнительная информация приведена в разделе Локальная сертификатная компания справочной системы Information Center.
  2. Создайте в созданной сертификатной компании сертификат системы.
  3. Укажите серверы, которые будут применять созданный сертификат системы.

Работа с сертификатами, выданными уполномоченной сертификатной компанией:

В IBM Toolbox for Java предусмотрен файл набора ключей, поддерживающий получение сертификатов серверов, выданных следующими компаниями:

- IBM World Registry
- Integrion Financial Network
- RSA Data Security, Inc.
- Thawte Consulting
- VeriSign, Inc.

Файл ключей уже поддерживает сертификаты, выданные перечисленными компаниями. Вы должны только получить файлы zip с алгоритмами шифрования и добавить их имена в переменную CLASSPATH.

Для применения сертификата выполните следующие действия:

1. Выберите каталог для хранения файлов zip.
2. Загрузите нужную версию SSL путем копирования в выбранный каталог следующих файлов:
   - Если вы планируете применять 56-разрядное шифрование (программу 5722-CE2), скопируйте файл /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip.
   - Если вы планируете применять 128-разрядное шифрование (лицензионную программу 5722-CE3), скопируйте файл /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip.
3. Добавьте путь к файлу zip в переменную CLASSPATH.

Работа с собственным сертификатом:

Если сертификаты уполномоченной сертификатной компании не применяются, необходимо загрузить собственные сертификаты сертификатной компании (CA) (со всех серверов, на которых они есть) для применения их с классами IBM Toolbox for Java. Кроме того, необходимо также получить файлы zip с алгоритмами шифрования и добавить их имена в переменную CLASSPATH.

Для применения собственных сертификатов выполните следующие действия:

1. Выберите каталог для хранения файлов zip.
2. Загрузите нужную версию SSL. Для этого необходимо скопировать как алгоритмы шифрования, так и утилиты для работы с собственными сертификатами:
   - Для 56-разрядного шифрования (применяемого в лицензионных программах 5722-CE2) скопируйте файлы /QIBM/ProdData/HTTP/Public/jt400/SSL56/sslightx.zip, cfwk.zip и ssltools.jar
   - Для 128-разрядного шифрования (применяемого в лицензионных программах 5722-CE3) скопируйте файлы /QIBM/ProdData/HTTP/Public/jt400/SSL128/sslightu.zip, cfwk.zip и ssltools.jar.
3. Добавьте имена файла ssltools.jar и файлов zip в переменную CLASSPATH.
4. Создайте в клиентской системе каталог <SSL>\com\ibm\as400\access, где <SSL> - это каталог, в который были скопированы файлы jar и zip.
5. В командной строке в каталоге <SSL> клиентской системы введите следующую команду:

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <имя_системы>:<порт>
   ```

   , где <порт> - это номер порта любого из серверов хоста. Например, защищенный сервер входа в систему iSeries по умолчанию работает с портом 9476.
6. Введите номер сертификата сертификатной компании (CA), который необходимо добавить в набор ключей. Убедитесь, что вы выбрали сертификат CA, а не сертификат сервера.
7. В качестве имени сертификата можно ввести любую строку букв и цифр.

   Примечание: Для добавления всех собственных сертификатов в класс KeyRing необходимо запустить экземпляры KeyringDB в каждой системе, содержащей такие сертификаты. Для добавления сертификатов запустите в каждой системе iSeries, в которой будут применяться соединения SSL, следующую команду:

   ```
   java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect <имя_системы>:<порт>
   ```

После выполнения указанных шагов настройка собственных сертификатов будет завершена. Перед запуском приложения убедитесь в том, что переменная CLASSPATH содержит следующие элементы:

- имя каталога, содержащего файл com\ibm\as400\access\KeyRing.class
- jt400.jar
- sslightx.zip или sslightu.zip (в зависимости от того, какой файл был загружен)

Поскольку файл jt400.jar содержит копию класса KeyRing.class по умолчанию, каталог с файлом com\ibm\as400\access\KeyRing.class должен быть указан в CLASSPATH перед файлом jt400.jar.

Примечание: Вместо внесения каталога, содержащего файл KeyRing.class, в переменную CLASSPATH можно заменить старый класс в файле jt400.jar новым классом KeyRing.class.

Применение SSL для шифрования данных, передаваемых между клиентом и сервером Proxy:

SSL позволяет шифровать данные, которыми обмениваются сервер и клиент Proxy. Для этого применяются файлы, поставляемые с лицензионной программой IBM iSeries Client Encryption (5722-CE2 или 5722-CE3). Как и в IBM Toolbox for Java, эти файлы представляют собой классы Java, не зависящие от платформы, которые позволяют клиенту и серверу Proxy работать в любой системе с виртуальной машиной Java.

Для шифрования данных, передаваемых между клиентом и сервером Proxy, выполните следующие действия:

1. Настройте сервер Proxy для поддержки шифрования данных.
2. Настройте клиент Proxy для поддержки шифрования данных.

3. Установите соединение с помощью объекта SecureAS400, для того чтобы продукт IBM Toolbox for Java выполнял шифрование данных.

Примечание: С помощью первых двух операций будет создано защищенное соединение между клиентом и сервером proxy. Для передачи по нему данных приложение должно работать с объектом SecureAS400. При работе с объектом AS400 данные не шифруются и передаются по обычному соединению.

Для шифрования данных, передаваемых между клиентом и сервером Proxy, требуются только классы Java, поставляемые с лицензионной программой Client Encryption (5722-CE2 или 5722-CE3). Для шифрования данных, передаваемых между сервером Proxy и сервером iSeries, необходимо дополнительно настроить эту функцию шифрования.

Настройка SSL на сервере Proxy:

Для работы с SSL у сервера Proxy должен быть сертификат. Сертификат сервера Proxy можно создать с помощью графического интерфейса IKeyman. Поскольку IKeyman - это программа с графическим интерфейсом, ее нужно запустить в системе клиента. После создания сертификата его можно скопировать в систему iSeries, если сервер Proxy работает в этой системе.

Для настройки сервера Proxy на обработку зашифрованных данных выполните следующие действия:

1. Настройте клиент для запуска программы IKeyman.
2. Создайте сертификат сервера Proxy.
3. Запустите сервер Proxy с созданным сертификатом.

Настройка графического интерфейса IKeyman в клиентской системе

IKeyman - это программа на Java, применяющая интерфейсы Java Swing 1.1. Для работы с IKeyman в клиентской системе должна бать установлена JVM с поддержкой Java 2 Standard Edition (J2SE).

Программа IKeyman входит в состав лицензионной программы IBM iSeries Client Encryption (5722-CE2 или 5722-CE3) и находится в файле ssltools.jar. Процедура настройки клиента для работы с SSL (и запуска IKeyman) зависит от версии применяемой лицензионной программы.

Настройте клиент для работы с SSL, выполнив следующие действия:

1. Выберите каталог рабочей станции, в котором будут находиться файлы jar и zip.
2. Скопируйте необходимые файлы в выбранный каталог:
   - Если вы планируете применять 56-разрядное шифрование, после загрузки программы 5722-CE2 в систему iSeries скопируйте на рабочую станцию следующие файлы:
     - /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip
     - /QIBM/ProdData/http/public/jt400/ssl56/cfwk.sec
   - Если вы планируете применять 128-разрядное шифрование, после загрузки лицензионной программы 5722-CE3 в систему iSeries скопируйте на рабочую станцию следующие файлы:
     - /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip
     - /QIBM/ProdData/http/public/jt400/ssl128/cfwk.sec
3. Добавьте имена файлов jar и zip в переменную CLASSPATH. Не добавляйте в CLASSPATH имя файла .sec.

   Примечание: Файл cfwk.zip должен быть указан в начале переменной CLASSPATH.

Создание сертификата сервера

Создайте собственный сертификат с помощью программы IKeyman.

Примечание: Если работа графического интерфейса IKeyman GUI завершается преждевременно, убедитесь, что файл cfwk.zip указан первым в переменной CLASSPATH и файл cfwk.sec находится в том же каталоге, что и cfwk.zip.

Создайте сертификат сервера Proxy, выполнив следующие действия:

1. Запустите программу IKeyman командой:

   ```
   java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
   ```

2. В меню **Файл базы данных ключей** программы IKeyman выберите пункт **Создать**.
3. В окне **Создать** не изменяйте **Тип базы данных ключей**, который должен быть равен **Класс базы данных ключей SSLight**.
4. Введите **Имя файла** (например, `ProxyServerKeyring.class`) или нажмите кнопку **Обзор** для выбора файла класса, который будет применяться в качестве файла ключей.

   Примечание: Запомните или запишите имя файла набора ключей, поскольку оно необходимо для запуска сервера proxy.
5. Введите **Расположение** (путь) или оставьте значение по умолчанию (имя текущего каталога) и нажмите **OK**.
6. В окне **Ввод пароля** заполните поля **Пароль** и **Подтверждение пароля**, а затем нажмите **OK**. (Опция **Задать срок действия** необязательна.)

   Примечание: Запомните свой пароль, поскольку он необходим для запуска сервера proxy. Значки ключей в этом окне показывают относительную длину пароля. Пароль будет надежнее, если он будет содержать цифры и буквы обоих регистров.
7. В меню **Создать** программы IKeyman выберите пункт **Создать собственный сертификат**.
8. В окне диалога **Создать собственный сертификат** заполните поля **Метка ключа** (например, `MyCertificate`) и **Организация**.
9. Щелкните по списку **Страны** и выберите страну или регион, введите **Период действия** или оставьте значение по умолчанию, затем нажмите **OK**.
10. В меню **Файл базы данных ключей** выберите **Закрыть**, а затем (в том же меню) - **Выход**.

После этого в текущем каталоге должен появиться созданный файл ключей.

Запуск сервера proxy с помощью нового сертификата

Перед запуском сервера Proxy убедитесь в том, что переменная CLASSPATH сервера содержит имена jt400.jar, sslightx.zip и имя каталога файла ключей.

Запустите сервер Proxy с созданным сертификатом. Укажите необходимые значения в параметрах -keyringName и -keyringPassword. Например:

```
java com.ibm.as400.access.ProxyServer -keyringName ProxyServerKeyring -keyringPassword pxypswrd
```

Настройка SSL на клиенте Proxy:

Ниже описана процедура добавления сертификата сервера в базу данных сертификатов клиента, хранящуюся в файле .class Java. Эту процедуру необходимо выполнить в том случае, если сервер применяет собственный сертификат.

Для настройки клиента Proxy на обмен зашифрованными данными выполните следующие действия:

1. Настройте сервер Proxy для поддержки шифрования данных, а затем запустите этот сервер.

2. Настройте SSL в системе клиента.
3. С помощью программы KeyringDB получите сертификат сервера Proxy.
4. Установите в системе клиента обновленный файл KeyRing.class.
5. Настройте параметры защиты Proxy в системе клиента.

Настройка SSL в клиентской системе

Средство для загрузки сертификата (KeyringDB) представляет собой программу на Java. KeyringDB входит в состав лицензионной программы IBM iSeries Client Encryption (5722-CE2 или 5722-CE3) и находится в файле ssltools.jar. Процедура настройки клиента для работы с SSL зависит от версии применяемой лицензионной программы.

После настройки сервера Proxy настройте клиент для работы с SSL, выполнив следующие действия:

1. Выберите каталог рабочей станции, в котором будут находиться файлы jar и zip.
2. Скопируйте необходимые файлы в выбранный каталог:
   - Если вы планируете применять 56-разрядное шифрование, после загрузки лицензионной программы 5722-CE2 в систему iSeries скопируйте на рабочую станцию следующие файлы:
     – /QIBM/ProdData/http/public/jt400/ssl56/sslightx.zip
     – /QIBM/ProdData/http/public/jt400/ssl56/ssltools.jar
     – /QIBM/ProdData/http/public/jt400/ssl56/cfwk.zip
   - Если вы планируете применять 128-разрядное шифрование, после загрузки лицензионной программы 5722-CE3 на сервер скопируйте на рабочую станцию следующие файлы:
     – /QIBM/ProdData/http/public/jt400/ssl128/sslightu.zip
     – /QIBM/ProdData/http/public/jt400/ssl128/ssltools.jar
     – /QIBM/ProdData/http/public/jt400/ssl128/cfwk.zip
3. Добавьте имена файлов jar и zip в переменную CLASSPATH.
4. Создайте в клиентской системе каталог <SSL>\com\ibm\as400\access, где <SSL> - это каталог, в который были скопированы файлы jar и zip.

Добавление сертификата сервера proxy

Программа KeyringDB создает новый файл KeyRing.class, содержащий сертификат сервера, и помещает его в подкаталог com\ibm\as400\access текущего каталога.

Добавьте сертификат сервера в файл KeyRing.class с помощью программы KeyringDB, выполнив следующие действия:

1. Перейдите в каталог с файлами jar и zip и вызовите следующую команду:

```
     java utilities.KeyringDB
com.ibm.as400.access.KeyRing -connect
Имя_сервера_proxy:порт
```

   где:

   - *Имя_сервера_proxy* - имя хоста сервера Proxy
   - *порт* - порт сервера Proxy (по умолчанию 3471)

     Например:

```
java utilities.KeyringDB com.ibm.as400.access.KeyRing -connect myProxyServer:3471
```

2. Когда система предложит выбрать сертификат, выберите сертификат 0.
3. В качестве имени сертификата можно ввести любую строку букв и цифр.

Применение в клиентской системе обновленного файла KeyRing.class

Файл jt400Proxy.jar содержит класс KeyRing.class. Для установки в системе клиента обновленного файла KeyRing.class убедитесь в том, что переменная CLASSPATH содержит:

- имя каталога, содержащего файл com\ibm\as400\access\KeyRing.class
- jt400Proxy.jar
- sslightx.zip или sslightu.zip (в зависимости от того, какой файл был загружен)
- cfwk.zip

Поскольку файл jt400Proxy.jar содержит копию класса KeyRing.class по умолчанию, каталог с файлом com\ibm\as400\access\KeyRing.class должен быть указан в CLASSPATH перед файлом jt400Proxy.jar.

Примечание: Вместо внесения каталога, содержащего файл KeyRing.class, в переменную CLASSPATH можно добавить в файл jt400Proxy.jar новый класс KeyRing.class.При этом будет удалена старая версия этого класса.

Настройка параметров защищенного сервера proxy в клиентской системе

Для того чтобы клиент Proxy подключался к серверу Proxy через защищенное соединение, задайте следующие параметры системы:

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

где *proxy-сервер* - имя хоста сервера Proxy

```
com.ibm.as400.access.SecureAS400.proxyEncryptionMode=режим
```

где *режим* - одно из следующих чисел:

- 1 - для шифрования данных, передаваемых между клиентом и сервером Proxy
- 2 - для шифрования данных, передаваемых между сервером Proxy и сервером iSeries
- 3 - для шифрования данных, передаваемых как между клиентом и сервером Proxy, так и между сервером Proxy и сервером iSeries

  Ниже приведен пример запуска приложения с применением SSL:

```
java -Dcom.ibm.as400.access.AS400.proxyServer=myProxyServer
   -Dcom.ibm.as400.access.SecureAS400.proxyEncryptionMode=1   myApplication
```

Службы идентификации

В IBM Toolbox для Java предусмотрены классы, взаимодействующие со службами защиты OS/400. В частности, поддерживается идентификация пользователей, или *субъектов*, путем сравнения их имен и паролей со значениями, заданными при регистрации в OS/400. После идентификации пользователю может быть выдано одноразовое разрешение. С помощью этого разрешения можно изменить владельца текущей нити OS/400, переключившись на идентифицированного пользователя. С точки зрения работы нити, такая смена владельца эквивалентна входу пользователя в систему.

Примечание: Службы создания разрешений и обмена ими поддерживаются только серверами выпуска V5R1M0 или выше.

Предоставляемая поддержка

В объекте AS400 предусмотрена функция идентификации пользователя по паролю на сервере. 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 Базовая служба защиты Java (JGSS). Дополнительная информация о JGSS приведена в разделе Документация по защите J2SDK версии 1.4 .

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.

Для применения разрешений необходимо получить экземпляры класса ProfileTokenCredential с помощью методов getProfileToken(). 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 OS/400. 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 OS/400 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.

Дополнительная информация приведена в следующих разделах справочной документации по Java:

Класс ProfileTokenCredential

Пример

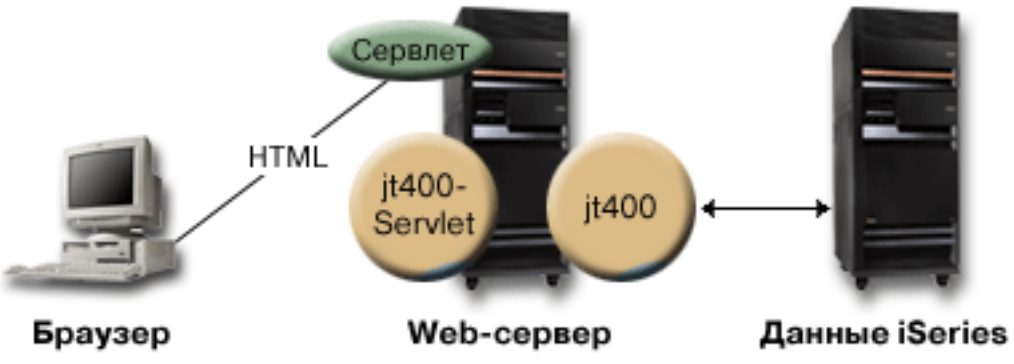
Ознакомьтесь с примером кода, в котором профайлу пользователя выдается одноразовое разрешение, позволяющее заменить владельца нити OS/400 и работать в системе от имени указанного пользователя.

Классы сервлетов

Классы сервлетов, поставляемые с IBM Toolbox for Java, применяются совместно с классами доступа, расположенными на Web-сервере, для предоставления пользователям информации, хранящейся на сервере iSeries. Вы можете применять классы сервлетов по своему усмотрению в разрабатываемых проектах.

На следующем рисунке показано, какую роль классы сервлетов играют в передаче данных iSeries от Web-сервера в браузер. 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. Web-сервер подключен к системе iSeries, в которой хранятся данные.

Рисунок 1: Принцип работы сервлетов



“Подробное описание рисунка 1: Работа сервлетов (rzahh585.gif)” на стр. 235

В IBM Toolbox for Java предусмотрено четыре типа классов сервлетов:

- Классы идентификации
- Классы RowData
- Классы RowMetaData
- Классы Converter

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)

IBM Toolbox for Java: Классы сервлетов

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.
- Изображения справа сервера iSeries, отмеченного надписью ’Данные iSeries’ и представляющего расположение данных, с которыми работает сервлет.
- Изображения в центре сервера iSeries, отмеченного надписью ’Web-сервер’ и представляющего Web-сервер. 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: Сервер WebServer может быть расположен в системе iSeries или вне ее; он может быть размещен даже в системе, указанной в образе iSeries Data.
- 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 Toolbox for Java (желтые круги): jt400Servlet.jar и jt400.jar. Классы в этих файлах поддерживают работу на Web-сервере сервлета, обменивающегося данными с сервером iSeries (изображение справа). Соответствующее соединение обозначено двунаправленной стрелкой.

Классы идентификации

Для реализации функции идентификации в пакете сервлета предусмотрены классы AuthenticationServlet и AS400Servlet.

Класс AuthenticationServlet

AuthenticationServlet - это реализация HttpServlet, выполняющая базовую идентификацию в сервлете. В подклассах класса AuthenticationServlet должен быть переопределен один или несколько следующих методов:

- validateAuthority() для выполнения идентификации (обязательно)
- bypassAuthentication() для фильтрации идентифицируемых запросов
- postValidation() для обработки запроса после идентификации

Методы класса AuthenticationServlet позволяют:
- Инициализировать сервлет
- Получить идентификатор идентифицированного пользователя
- Задавать идентификатор пользователя после пропуска идентификации
- Заносить в протокол исключительные ситуации или сообщения

Класс AS400Servlet

Класс AS400Servlet - это абстрактный подкласс класса AuthenticationServlet, представляющий сервлет HTML. Для совместного использования соединений и управления числом соединений пользователя сервлета с сервером применяется пул соединений.

Методы класса AS400Servlet позволяют:
- Проверять права доступа пользователя (путем переопределения метода validateAuthority() класса AuthenticationServlet)
- Подключаться к системе
- Получать и возвращать объекты пула соединений
- Закрывать пул соединений
- Получать и задавать теги заголовка документа HTML
- Получать и задавать теги завершения документа HTML

Дополнительные источники информации о сервлетах перечислены в разделе ссылок на справочную информацию.

Класс RowData

RowData - это абстрактный класс, предоставляющий способ описания и установки доступа к списку данных.

Существует четыре основных класса, расширяющих класс RowData:
- Класс ListRowData
- Класс RecordListRowData
- Класс ResourceListRowData
- Класс SQLResultSetRowData

Классы RowData позволяют:
- Получать и устанавливать текущую позицию
- Получать данные из ячейки указанного столбца с помощью метода getObject()
- Получать метаданные для строки
- Получать (get) или задавать (set) свойства объекта в данном столбце
- Определять число строк в списке с помощью метода length().

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()

Класс ListRowData:

Класс ListRowData позволяет выполнить следующие операции:

- Добавить или удалить строку из таблицы результатов.
- Получить и задать строку
- Получить информацию о столбцах таблицы с помощью метода getMetaData()
- Задать параметры столбцов с помощью метода setMetaData()

Класс ListRowData представляет список данных. ListRowData позволяет с помощью IBM Toolbox for Java “Классы доступа” на стр. 23 представлять данные различных типов, включая данные:

- 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. 640

Класс RecordListRowData:

Класс RecordListRowData позволяет выполнить следующие операции:

- Добавлять и удалять строки в списке записей.
- Получать и задавать строки
- Задавать формат записи с помощью метода setRecordFormat
- Получать формат записи.

Класс RecordListRowData предназначен для работы со списком записей. Запись можно получить с сервера в одном из следующих форматов:

- Как запись файла сервера
- Как запись очереди данных
- Как значение параметра в вызове программы
- 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.

Класс ResourceListRowData:

Класс ResourceListRowData представляет список ресурсов данных. Объект ResourceListRowData представляет реализацию интерфейса ResourceList.

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:
- Получать и задавать ИД атрибутов столбцов
- Получать и задавать списки ресурсов
- Определять число строк в списке
- Получать значения столбцов в текущей строке
- Получать списки свойств объекта данных
- Получать метаданные списка

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:

> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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 SQLResultSetRowData:

Класс SQLResultSetRowData представляет результат обработки запроса SQL в виде списка данных. Эти данные формируются в ходе обработки оператора SQL с помощью JDBC. С помощью методов этого класса можно получать и задавать метаданные набора результатов.

В приведенном примере проиллюстрировано применение классов ListRowData и HTMLTableConverter. Он включает программу на Java, соответствующий код HTML и пример Web-страницы.

Классы RowMetaData

Класс RowMetaData определяет интерфейс, применяемый для поиска информации о столбцах объекта RowData.

Классы RowMetaData позволяют выполнять следующие операции:
- Получать число столбцов
- Получать имя (name), тип (type) и (size) столбца
- Получать (get) или задавать (set) метку столбца
- Получать значения точности (precision) и числа десятичных знаков (scale) данных столбца
- Определять, являются ли данные столбца текстом (text)

Класс RowMetaData содержит три основных класса, реализующих, помимо своих собственных функций, все вышеперечисленные функции RowMetaData:

- Класс ListMetaData
- Класс RecordFormatMetaData
- Класс SQLResultSetMetaData

Класс ListMetaData:

Класс ListMetaData позволяет просматривать и изменять параметры столбцов класса “Класс ListRowData” на стр. 237. Для задания числа столбцов применяется метод setColumns(), удаляющий старое значение. 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. 640

Класс RecordFormatMetaData:

RecordFormatMetaData использует класс RecordFormat IBM Toolbox for Java. Он позволяет задавать формат записи в параметрах конструктора или использовать для доступа к формату записи методы get и set.

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));
}
```

Класс SQLResultSetMetaData:

Класс SQLResultSetMetaData хранит информацию о столбцах объекта SQLResultSetRowData. Набор результатов можно задать в конструкторе. Для получения и изменения метаданных набора результатов предназначены методы get и set.

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));
   }
}
```

Классы преобразования

Классы преобразования применяются для преобразования строк информации в массивы форматированных строк. Результатом будет представление в формате HTML, готовое к использованию на странице HTML. Преобразование выполняется следующими классами:

- StringConverter
- HTMLFormConverter
- HTMLTableConverter

Класс StringConverter:

Класс StringConverter - это абстрактный класс, описывающий преобразование строк данных. Собственно преобразование выполняется методом convert(). Этот метод возвращает одномерный массив, в который была преобразована строка данных.

Класс HTMLFormConverter:

Класс HTMLFormConverter расширяет класс StringConverter за счет дополнительного метода преобразования - 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:

- Выравнивание
- Расстояние между ячейками
- Ссылки в заголовках
- Ширина

Пример: Применение класса HTMLFormConverter

Ниже приведен пример использования класса HTMLFormConverter. (Этот пример можно откомпилировать и запустить на работающем Web-сервере.)

Применение класса HTMLFormConverter

Класс HTMLTableConverter:

Класс HTMLTableConverter расширяет класс StringConverter, добавляя новый метод convertToTables(). Этот метод служит для преобразования строковых данных в массив таблиц HTML, которые сервлет может применять для вывода списка в браузере.

Методы getTable() и setTable() позволяют задать для преобразования таблицу по умолчанию. Заголовки таблицы могут задаваться в объекте таблицы HTML или в мета-данных заголовка HTML, если метод setUseMetaData() вызван с параметром true.

Метод setMaximumTableSize() позволяет ограничить число строк в таблице. 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:

> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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 Utility

Классы Utility предназначены для выполнения общих задач. В IBM Toolbox for Java предусмотрены следующие утилиты:

- AS400ToolboxInstaller: предназначен для установки и обновления классов IBM Toolbox for Java на клиенте. Эта функция доступна как в виде программы на Java, так и в виде API.
- AS400ToolboxJarMaker: ускоряет загрузку файла JAR IBM Toolbox for Java путем создания из него файла JAR меньшего размера или распаковки отдельных файлов.
- CommandHelpRetriever: получает и создает текст справки для команд управляющего языка OS/400 (CL).
- CommandPrompter: запрашивает у пользователя параметры для выполнения конкретной команды. Действие класса CommandPrompter схоже с результатом нажатия клавиши F4 в командной строке iSeries и с командной строкой централизованного управления.
- RunJavaApplication и VRunJavaApplication: позволяют запускать программы на Java на сервере iSeries из командной строки.
- JPing: позволяет узнать, какие службы сервера активны. Кроме того, можно проверить отклик портов SSL.

Установка и обновление классов на клиенте

К классам IBM Toolbox for Java можно обращаться, указывая их расположение в интегрированной файловой системе сервера. Так как к этому расположению применяются временные исправления программ (PTF), программы на Java, обращающиеся к классам непосредственно на сервере, получают обновления автоматически. В некоторых случаях обращаться к классам на сервере не удобно, например:

- Если рабочая станция подключена к серверу по линии связи с низким быстродействием, то загрузка классов с сервера на рабочую станцию будет выполняться слишком медленно.
- Если приложения на Java используют переменную CLASSPATH для доступа к классам в клиентской файловой системе, то для перенаправления вызовов на сервер iSeries требуется программа iSeries Access для Windows. В некоторых случаях эту программу нельзя установить в клиентской системе.

В перечисленных выше случаях оптимальным вариантом является установка классов в клиентской системе. Класс AS400ToolboxInstaller предоставляет функции установки и обновления классов IBM Toolbox for Java в клиентской системе.

Применение AS400ToolboxInstaller

Объект AS400ToolboxInstaller является программой и программируемым интерфейсом. Этот объект содержит метод main(), поэтому его можно запустить из командной строки. Кроме того, в нем предусмотрены общие дополнительные методы, поэтому его можно добавить в приложение и вызвать из него.

Объект AS400ToolboxInstaller позволяет установить файлы IBM Toolbox for Java на клиенте, а затем обновлять их по мере необходимости. При первом запуске файлы Toolbox копируются с сервера на рабочую станцию. После применения PTF на сервере объект AS400ToolboxInstaller обновляет файлы на рабочей станции, заново загружая их с сервера.

Класс AS400ToolboxInstaller копирует файлы в локальную файловую систему клиента. Многие браузеры запрещают аплетам записывать файлы в локальную файловую систему, поэтому этот класс может не работать вместе с аплетами.

Запуск класса AS400ToolboxInstaller из командной строки

Класс AS400ToolboxInstaller можно вызывать как независимую программу из командной строки. Это позволяет избежать разработки специальной программы. Вместо этого вы можете запустить класс как приложение на Java, которое установит, удалит или обновит классы IBM Toolbox for Java.

Вызовите класс AS400ToolboxInstaller с помощью следующей команды, указав опцию установки, удаления или сравнения:

```
java utilities.AS400ToolboxInstaller [опции]
```

Опция **-source** указывает расположение классов IBM Toolbox for Java, а опция **-target** - целевой каталог для размещения классов IBM Toolbox for Java в клиентской системе.

Кроме этого, с помощью опций можно выбрать установку всего пакета или отдельных функций. Например, установить и обновить классы пакета доступа IBM Toolbox for Java (jt400.jar) на рабочей станции можно с помощью следующей команды:

```
java utilities.AS400ToolboxInstaller -install -package ACCESS -source myAS400    -target c:\toolbox
```

В приведенном выше примере предполагается, что файлы .jar продукта IBM Toolbox for Java находятся в каталоге c:\toolbox.

Добавление класса AS400ToolboxInstaller в программу

Класс AS400ToolboxInstaller предоставляет интерфейсы прикладных программ (API), необходимые для установки, удаления и обновления классов IBM Toolbox for Java из программы, работающей в клиентской системе.

Для установки и обновления классов IBM Toolbox for Java служит метод install(). При установке и обновлении требуется указать в программе исходный и целевой пути, а также имена пакетов классов. Исходный URL указывает расположение управляющих файлов на сервере. Структура каталогов копируется с сервера в клиентскую систему.

Метод install() выполняет только копирование файлов. Он **не** обновляет переменную среды CLASSPATH. Если метод install() был выполнен успешно, программа на Java может вызвать метод getClasspathAdditions() для того чтобы узнать, какие параметры следует добавить в переменную среды CLASSPATH.

Ниже приведен пример установки файлов с сервера ″mySystem″ в каталог ″jt400″ на диске d: с помощью класса AS400ToolboxInstaller, а также описано, как можно определить данные, которые необходимо добавить в переменную среды CLASSPATH:

```
                    // Установка классов IBM Toolbox for Java
                    // в клиентской системе.
URL sourceURL = new URL("http://mySystem.myCompany.com/QIBM/ProdData/HTTP/Public/jt400/");

if (AS400ToolboxInstaller.install(
        "ACCESS",
        "d:\\jt400",
        sourceURL))

{
                    // Если классы IBM Toolbox for Java были
                    // установлены или обновлены, то - определение,
                    // что требуется добавить в переменную CLASSPATH.
   Vector additions = AS400ToolboxInstaller.getClasspathAdditions();

                    // Если CLASSPATH необходимо обновить,
   if (additions.size() > 0)
   {
                    // ... то - обработка всех добавлений в CLASSPATH.
   }
}

                    // ... В противном случае, обновлять переменную не требуется.
```

С помощь метода isInstalled() можно узнать, установлены ли в клиентской системе классы IBM Toolbox for Java. Применение метода isInstalled() позволяет решить, следует ли выполнить установку немедленно или ее можно отложить.

Метод install() устанавливает и обновляет файлы в системе. Программа на Java может вызвать метод isUpdateNeeded(), для того чтобы определить, требуется ли выполнить обновление с помощью метода install().

Метод unInstall() позволяет удалить классы IBM Toolbox for Java из клиентской системы. Метод unInstall выполняет только удаление файлов; переменная CLASSPATH при этом не изменяется. Для того чтобы узнать, какие переменные следует удалить из переменной среды CLASSPATH, вызовите метод getClasspathRemovals().

Дополнительные примеры установки и обновления классов из клиентской программы с помощью класса AS400ToolboxInstaller приведены в примере Установка и обновление.

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.

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.
- Класс AS400ToolboxJarMaker представляет собой расширенную версию инструмента JarMaker, адаптированную для работы с файлами JAR IBM Toolbox for Java.

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

Полный список опций, которые можно указывать в командной строке, приведен в следующих разделах:

- Опции базового класса JarMaker
- Дополнительные опции подкласса AS00ToolboxJarMaker

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
- Исключать из файла JAR все файлы IBM Toolbox for Java, которые не требуются для работы приложений

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:

- Текущий каталог (extract(файл-jar))
- Другой каталог (extract(файл-jar, каталог))

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. Для этого в классе AS400ToolboxJarMaker предусмотрена функция split(файл-jar, максимальный-размер).

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

С помощью AS400ToolboxJarMaker из файла JAR можно удалить не используемые приложением файлы IBM Toolbox for Java, оставив только необходимые для работы приложения компоненты, языки и идентификаторы CCSID. Кроме того, AS400ToolboxJarMaker позволяет добавлять и удалять файлы JavaBean, связанные с выбранными компонентами.

Например, следующая команда создает файл JAR, содержащий только классы IBM Toolbox for Java, необходимые для работы компонентов CommandCall и ProgramCall:

```
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.

Компоненты, поддерживаемые 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
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

Компонент	Ключевое слово	Константа
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
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

Кодировки и CCSID, поддерживаемые IBM Toolbox for Java:

Продукт IBM Toolbox for Java поставляется с набором таблиц преобразования, имена которых совпадают с CCSID. Эти таблицы используются внутренними классами IBM Toolbox for Java (такими как CharConverter) для преобразования данных при обмене информацией с сервером iSeries. Например, таблица преобразования для CCSID 1027 находится в файле com/ibm/as400/access/ConvTable1027.class. Таблицы преобразования для следующих CCSID содержатся в файле jar IBM Toolbox for Java; поддержка других кодировок обеспечивается благодаря JDK. Таблицы преобразования больше не загружаются с сервера. Все CCSID, для которых не может быть найдена таблица преобразования или кодировка JDK, вызовут исключительную ситуацию. Некоторые из перечисленных таблиц могут совпадать с таблицами, включенными в JDK. В настоящий момент IBM Toolbox for Java поддерживает 122 значения CCSID OS/400.

Дополнительная информация о CCSID, включая полный список CCSID, поддерживаемых серверами iSeries, приведена в разделе Глобализация.

CCSID, поддерживаемые 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)
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

CCSID	Формат	Описание
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)
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

CCSID	Формат	Описание
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, латиница-2
1251	ASCII/ISO/Windows	Windows, кириллица
1252	ASCII/ISO/Windows	Windows, латиница-1
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, Балтика
1258	ASCII/ISO/Windows	Windows, Вьетнам
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)

CCSID	Формат	Описание
5026	Смешанный EBCDIC	Японский
5035	Смешанный EBCDIC	Японский
5123	Однобайтовый EBCDIC	Японский (только однобайтовый, с поддержкой евро)
5351	ASCII/ISO/Windows (двунаправленный)	Windows, иврит ST5 (с поддержкой евро)
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	Unicode iSeries (в основном используется в IFS)
62211	Однобайтовый EBCDIC	Иврит EBCDIC ST5
62224	Однобайтовый EBCDIC	Арабский EBCDIC ST6
62235	Однобайтовый EBCDIC	Иврит EBCDIC ST6
62245	Однобайтовый EBCDIC	Иврит EBCDIC ST10

Класс CommandHelpRetriever

С помощью класса CommandHelpRetriever можно получать текст справки для команд управляющего языка OS/400 (CL) и представлять его в форматах HTML или Диспетчера пользовательского интерфейса (UIM). Класс CommandHelpRetriever можно запускать с помощью командной строки или вызывать в программе на Java.

Для применения CommandHelpRetriever на сервере должна быть установлена система OS/400 версии V5R1 или выше, а в переменной CLASSPATH должны быть указаны анализатор XML и обработчик XSL. Дополнительная информация приведена на следующей странице:

“Анализатор XML и обработчик XSLT” на стр. 412

Кроме того, класс CommandHelpRetriever применяется командой CL Создать документацию для команды (GENCMDDOC). Поэтому для применения функций класса CommandHelpRetriever можно воспользоваться командой GENCMDDOC. Дополнительная информация приведена на следующей странице:

Команда Создать документацию для команды (GENCMDDOC)

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:

- Библиотеку системы iSeries, в которой расположена команда CL. Команды системы размещены в библиотеке QSYS.
- Команду CL.

Можно также указать такие дополнительные параметры CommandHelpRetriever, как имя системы iSeries, ИД пользователя, пароль и расположение целевого файла.

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);
```

Справочная документация по Java

Дополнительная информация о классе CommandHelpRetriever приведена в следующем разделе справочной документации по Java:

Класс CommandHelpRetriever

Класс CommandPrompter

Класс CommandPrompter позволяет запросить параметр заданной команды. Класс CommandPrompter можно сравнить с приглашением команды CL iSeries (выдаваемым при нажатии F4) и приглашением команды в Централизованном управлении.

Для применения класса CommandPrompter необходимо, чтобы на сервере была установлена OS/400 версии V4R4 или выше. Дополнительная информация приведена в разделах Информационные отчеты APAR Навигатора iSeries и Необходимые исправления Graphical Command Prompter.

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. 412

Все эти файлы jar, кроме jhall.jar, входят в состав IBM Toolbox for Java. Дополнительная информация о файлах jar IBM Toolbox for Java приведена в разделе Файлы Jar. Дополнительные сведения о загрузке файла jhall.jar приведены на Web-сайте Sun JavaHelp(TM) .

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. 709

Класс RunJavaApplication

Классы RunJavaApplication и VRunJavaApplication позволяют запускать программы на Java в виртуальной машине Java системы iSeries. В отличие от классов JavaApplicationCall и VJavaApplicationCall, вызываемых из программ на Java, классы RunJavaApplication и VRunJavaApplication представляют собой полноценные программы.

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 она может получать входные данные через стандартный ввод. Результаты выполняемой программы 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.

Класс JPing

Класс JPing является утилитой командной строки, которая опрашивает серверы и составляет список активных служб и занятых портов. Отправить запрос на серверы из приложения на Java можно с помощью класса AS400JPing.

Дополнительная информация о применении этого класса в программах на Java приведена в разделе Документации по JPing.

Формат вызова JPing из командной строки:

```
java utilities.JPing система [опции]
```

где:

- система - имя сервера iSeries, которому необходимо отправить запрос
- [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*OS/400_Service***(-s***OS/400_Service***)**
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

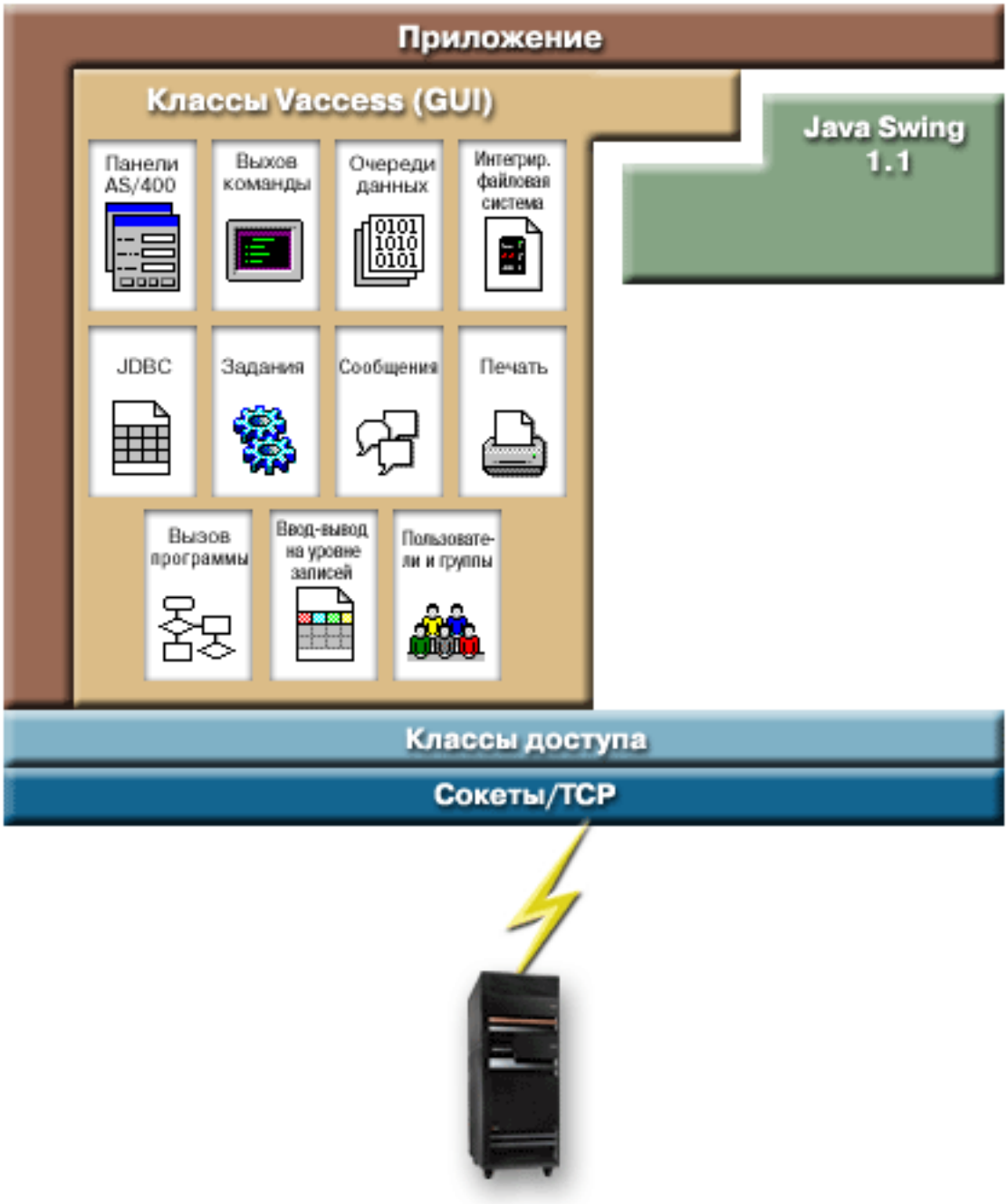
Ниже приведен пример вызова команды для опроса службы as-dtaq (в том числе портов SSL) с тайм-аутом 5 секунд:

```
java utilities.JPing myServer -s as-dtaq -ssl -t 5000
```

Классы Vaccess

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)” на стр. 255

Классы Vaccess

IBM Toolbox for Java содержит набор классов графического пользовательского интерфейса (GUI) в пакете vaccess. Эти классы являются классами доступа и предназначены для получения данных и представления их в удобном пользователю виде.

Для применения классов графического интерфейса из пакета IBM Toolbox для Java в программах на Java необходим продукт Swing 1.1, который входит в комплект поставки продукта Java 2 Platform, Standard Edition (J2SE). Дополнительная информация о Swing приведена на Web-сайте Sun Java Foundation Classes .

Дополнительная информация о взаимосвязи между классами GUI IBM Toolbox for Java, классами доступа и библиотекой Java Swing показана на Диаграмме классов Vaccess.

Для вывода данных iSeries применяются панели AS400.

Предусмотрены API для доступа к следующим ресурсам и средствам iSeries:
- Вызов команд
- Очереди данных
- События при ошибках*
- Интегрированная файловая система
- JavaApplicationCall
- JDBC
- Задания*
- Сообщения*
- Права доступа
- Средства печати*, включая программу просмотра буферных файлов
- Классы ProgramCall и ProgramParameter
- Доступ на уровне записей
- Списки ресурсов
- Состояние системы
- Системные значения
- Пользователи и группы

Примечание: Панели AS400 применяются с другими классами графического интерфейса (см. выше элементы, отмеченные звездочкой) для отображения и управления ресурсами iSeries.

При создании программ с применением компонентов GUI IBM Toolbox for Java рекомендуется обрабатывать ошибки и сообщать о них пользователю с помощью классов событий при ошибках.

Дополнительная информация о работе с данными системы iSeries приведена в разделе Классы доступа.

Подробное описание рисунка 1: Классы Vaccess (rzahh508.gif):

IBM Toolbox for Java: Диаграмма пакета классов графического интерфейса

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 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 доступа IBM Toolbox for Java.
- Расположенный под голубым прямоугольником синий прямоугольник такого же размера с именем ’Сокеты/TCP.’

- Изображение сервера iSeries в нижней части рисунка.
- Молнию, связывающую синий прямоугольник Сокеты/TCP с сервером, представляющую соединение через сокет между приложением на Java и сервером.

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

Приложение на Java применяет классы доступа IBM Toolbox for Java (голубой прямоугольник) для создания соединений между сокетами (синий прямоугольник). Эти соединения позволяют приложению на Java обмениваться данными (изображение молнии) с сервером (изображение сервера iSeries внизу).

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, поэтому их можно добавлять к любым фреймам, окнам и контейнерам AWT.

Предусмотрены следующие объекты AS400Pane:
- Класс AS400DetailsPane показывает список ресурсов сервера в виде таблицы, в каждой строке которой содержится информация об одном ресурсе. Пользователь может выбрать в таблице произвольное число ресурсов.
- Класс AS400ExplorerPane объединяет функции объектов AS400TreePane и AS400DetailsPane, показывая подробную информацию об объекте, выбранном в дереве ресурсов.
- Класс AS400JDBCDataSourcePane представляет значения свойств объекта AS400JDBCDataSource.
- Класс AS400ListPane представляет список ресурсов сервера, некоторые из которых можно выбрать.
- Класс AS400TreePane представляет иерархический список ресурсов сервера, некоторые из которых можно выбрать.

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 для 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.

Все ресурсы представляют собой реализацию интерфейса VNode.

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:

- На панели AS400ListPane выводится список дочерних объектов корневого объекта.
- На панели AS400DetailsPane выводится таблица с информацией о дочерних объектах корневого объекта.
- На панели AS400TreePane выводится дерево объектов, начиная с корневого объекта.
- На панели AS400ExplorerPane выводится дерево, начиная с корневого объекта.

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.
                    //
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.

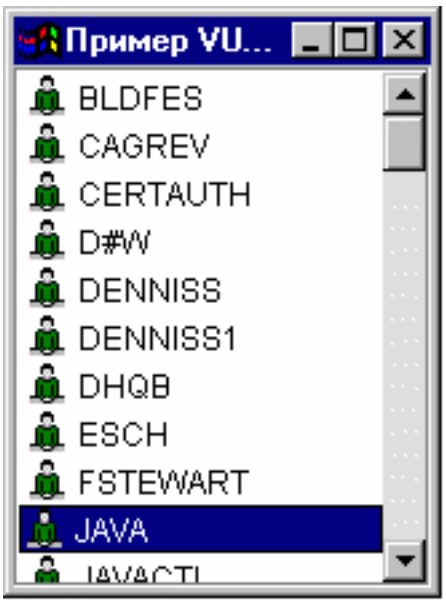
Предусмотрены следующие модели:

- 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

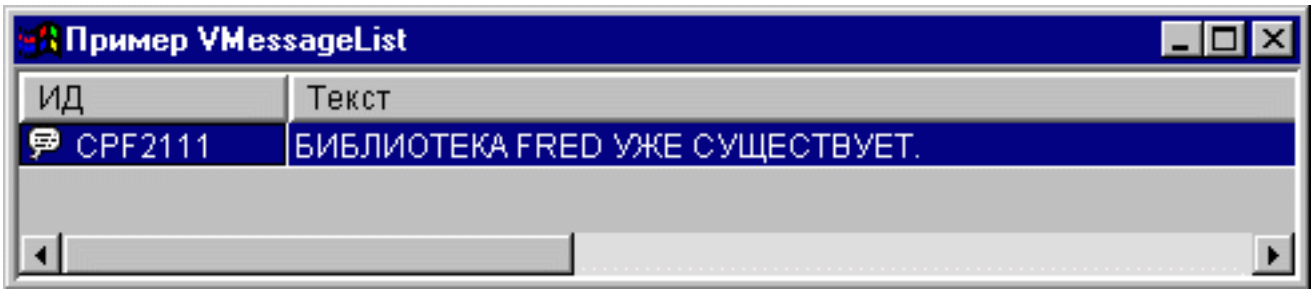
- Показывает список пользователей системы на панели AS400ListPane с помощью объекта VUserList. На рис. 1 показан окончательный результат:

  Рисунок 1: Применение панели AS400ListPane и объекта VUserList



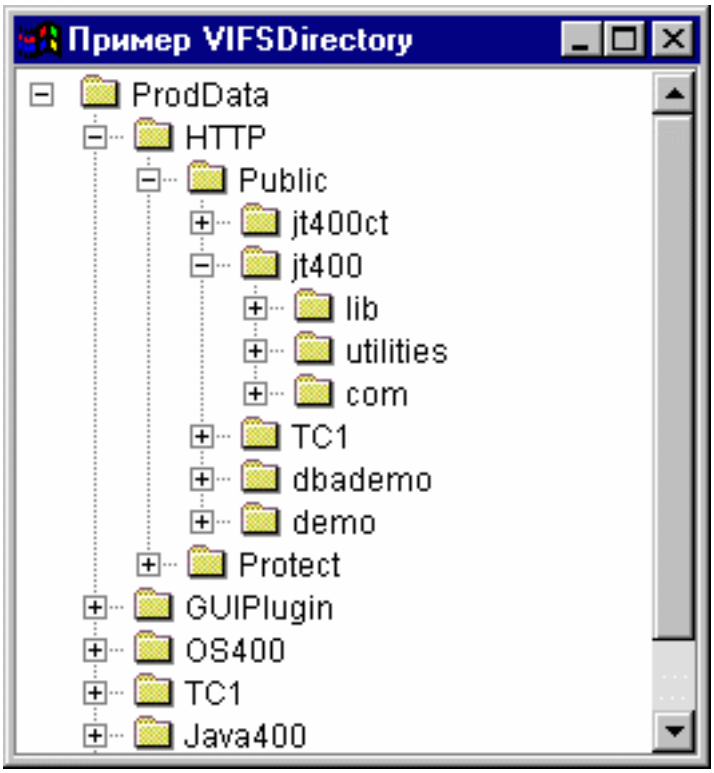
- Показывает список сообщений команды на панели AS400DetailsPane с помощью объекта VMessageList. На рис. 2 показан окончательный результат:

  Рисунок 2: Применение панели AS400DetailsPane и объекта VMessageList



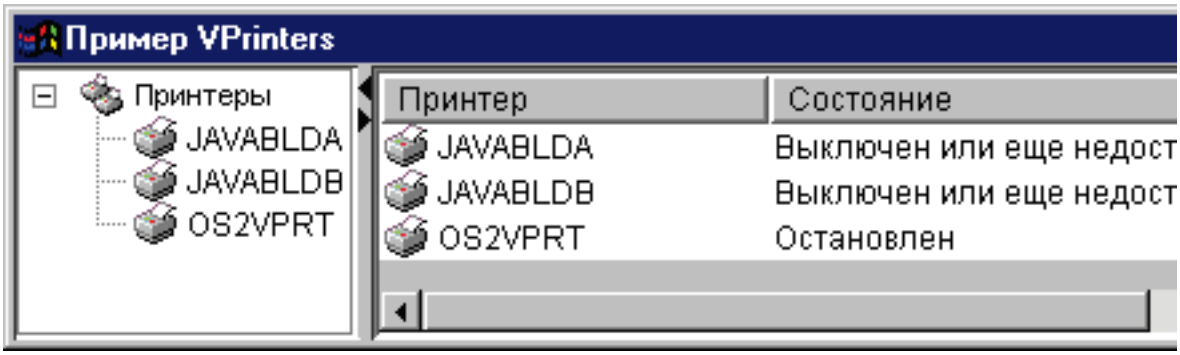
- Показывает дерево файлов интегрированной файловой системы на панели AS400TreePane с помощью объекта VIFSDirectory. На рис. 3 показан окончательный результат:

  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



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 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:

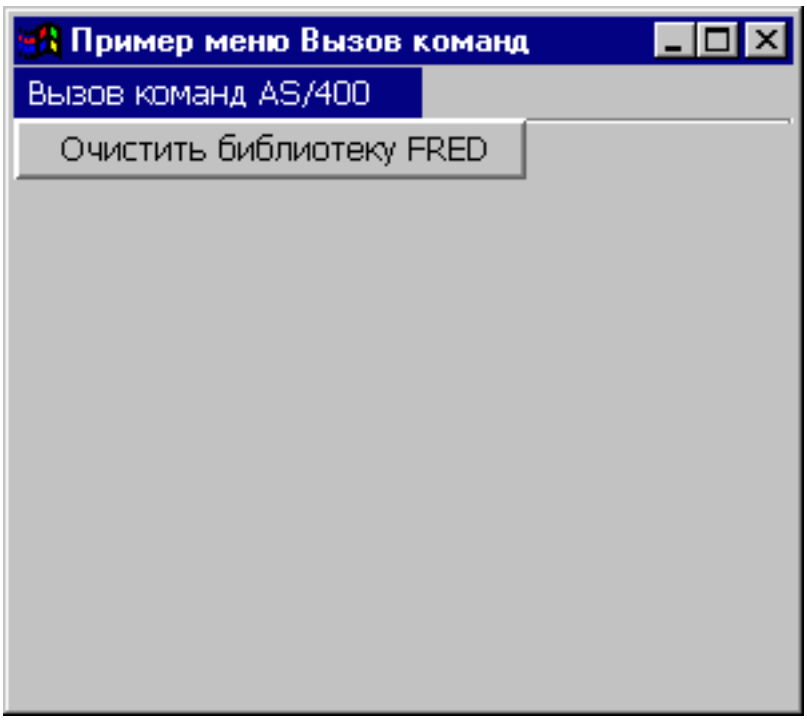
```
                    // Добавление обработчика ActionCompletedListener,
                    // реализованного с помощью анонимного
                    // внутреннего класса. Это наиболее удобный
                    // способ создания несложных обработчиков
                    // событий.
    button.addActionCompletedListener (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 JFC для обмена информацией с очередями данных сервера.

Классы DataQueueDocument и KeyedDataQueueDocument реализуют интерфейс Document JFC. 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);
```

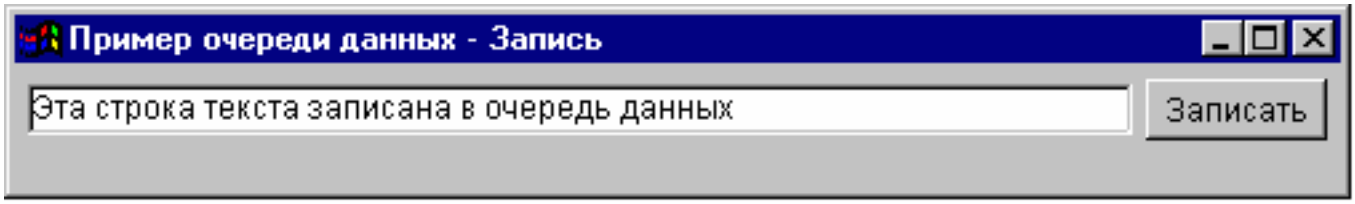
Первоначально текстовый компонент пуст. В него можно загрузить очередной элемент очереди данных с помощью метода read() или peek(). Метод write() позволяет записать содержимое текстового компонента в очередь данных. Учтите, что эти документы могут применяться только с элементами очередей данных типа 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



События при ошибках

В большинстве случаев, при возникновении ошибок компоненты графического пользовательского интерфейса (GUI) IBM Toolbox for Java генерируют соответствующее событие, а не исключительную ситуацию.

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.

В IBM Toolbox for Java предусмотрен компонент GUI ErrorDialogAdapter, который автоматически выводит на экран окно диалога при генерации события при ошибке.

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);
```

Интегрированная файловая система

С помощью компонентов GUI, предназначенных для работы с IFS, программы на Java могут показывать каталоги и файлы IFS сервера в стандартных окнах GUI.

Предусмотрены следующие компоненты:

- IFSFileSystemView позволяет работать с интегрированной файловой системой iSeries.
- Объект IFSFileDialog - это окно диалога, позволяющее просматривать содержимое каталогов и выбрать нужный файл.
- Объект VIFSDirectory - это ресурс, представляющий каталог IFS. Он рассчитан на применение с объектами AS400Pane.
- Объект IFSTextFileDocument представляет текстовый файл и может применяться с любым текстовым компонентом GUI JFC.
- 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).

Укажите другой путь вместо ″/QSYS.LIB″, поскольку этот каталог, как правило, содержит большой объем данных и их загрузка может занять много времени.

Класс IFSFileSystemView:

Класс IFSFileSystemView позволяет работать с интегрированной файловой системой iSeries при создании объектов javax.swing.JFileChooser.

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());
}
```

Класс VIFSFileDialog:

Класс IFSFileDialog представляет окно диалога, в котором можно просматривать содержимое каталогов IFS сервера и выбирать нужные файлы. Названия кнопок этого окна можно задать по своему усмотрению. Кроме того, объект FileFilter позволяет пользователю ограничить список доступных для выбора файлов.

Для получения имени выбранного файла применяется метод getFileName(). Метод getAbsolutePath() позволяет получить полное имя файла.

В следующем примере создается окно диалога выбора файла с двумя фильтрами:

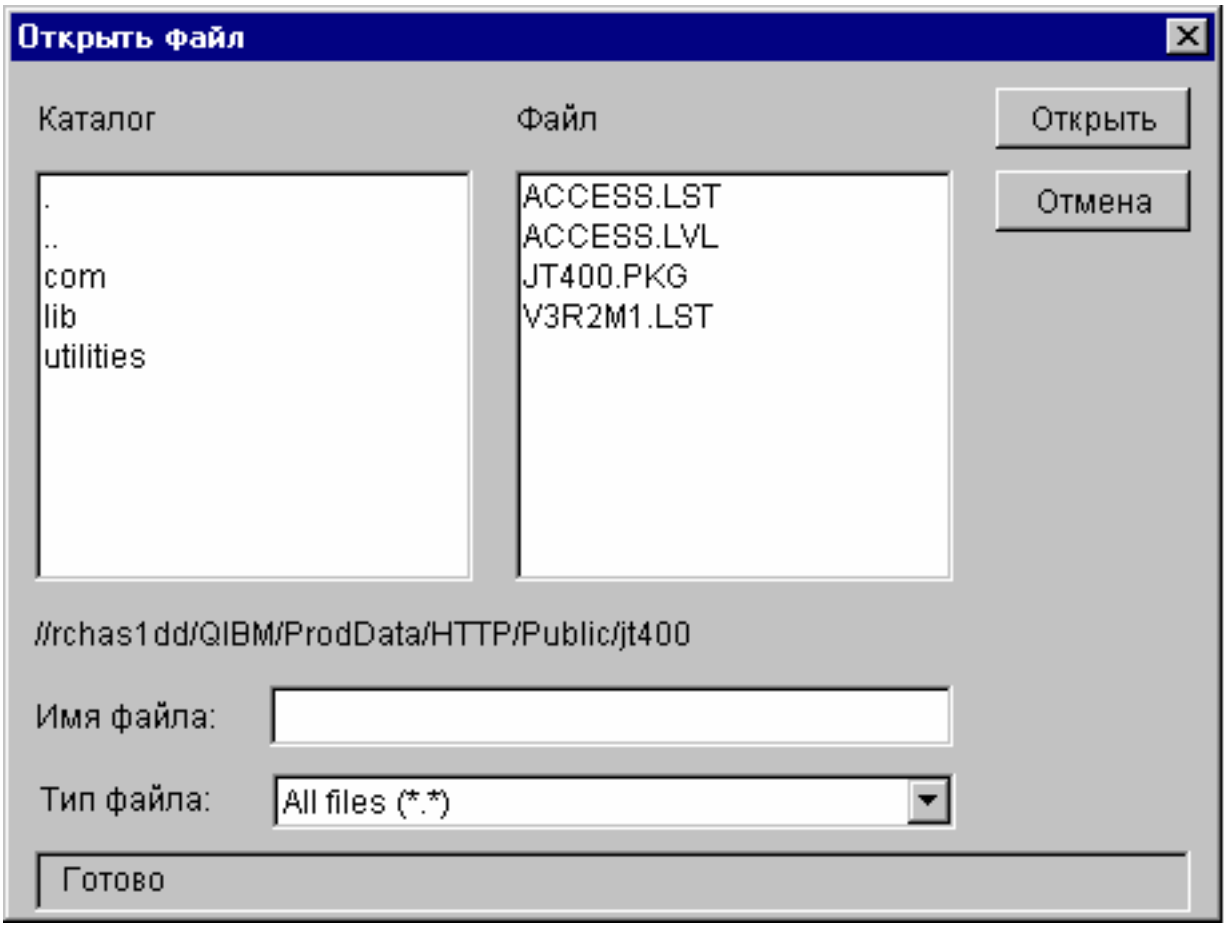
```
                    // Создание объекта IFSFileDialog
                    // с указанной строкой заголовка.
                    // Предполагается, что объект AS400
                    // и фрейм - это объекты JFrame,
                    // созданные и инициализированные отдельно.
IFSFileDialog dialog = 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



Каталоги в объектах AS400Pane:

Объекты AS400Pane - это компоненты GUI, предназначенные для работы с ресурсами сервера. Объект VIFSDirectory - это ресурс, представляющий каталог интегрированной файловой системы в объекте AS400Pane. 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 ″?″. Метод setFilter() позволяет указать в качестве фильтра объект IFSFileFilter.

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:

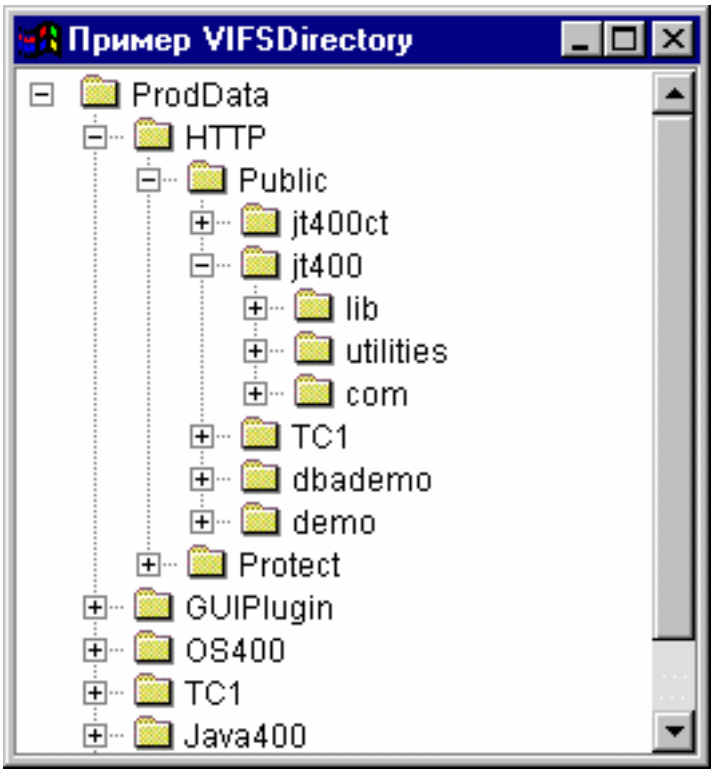
```
                    // Создание объекта VIFSDirectory.
                    // Предполагается, что "system" - это объект AS400, который
                    // уже создан и инициализирован.
                    //
VIFSDirectory root = new VIFSDirectory (system, "/DirectoryA/DirectoryB");

                    // 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 IFSTextFileDocument:

Документы текстовых файлов позволяют программам на Java применять любые текстовые компоненты GUI JFC для работы с текстовыми файлами, хранящимися в IFS сервера. (Текстовыми компонентами называются графические компоненты, предназначенные для вывода и редактирования текста).

Класс IFSTextFileDocument реализует интерфейс Document JFC. 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.
```

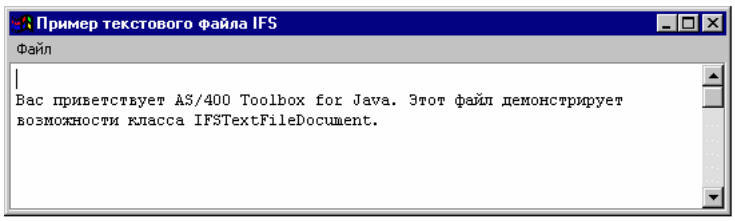
```
JTextArea textArea = new JTextArea (ifsDocument);

                   // Добавление текстовой области во фрейм.
                   // Предполагается, что фрейм JFrame
                   // уже создан.
frame.getContentPane ().add (textArea);
```

Пример

Представляет объект IFSTextFileDocument в панели JTextPane.

На рис. 1 показан компонент GUI IFSTextFileDocument:

Рисунок 1: Пример объекта IFSTextFileDocument



Класс VJavaApplicationCall

Класс VJavaApplicationCall позволяет запускать приложения на Java из клиентской системы на сервере с помощью графического пользовательского интерфейса (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 в стандартном потоке ввода. Дополнительная информация приведена в разделе Опции команд Java.

С помощью этого примера можно создать следующий GUI для программы на Java.

Класс VJavaApplicationCall - это класс, вызываемый из программы на Java. Кроме этого, в IBM Toolbox for Java присутствует утилита, которую можно применять для вызова программы на Java с рабочей станции. Эта утилита представляет собой полное приложение на Java. Дополнительная информация приведена в разделе Класс RunJavaApplication.

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 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

Объект SQLConnection представляет в JDBC соединение с базой данных. **Объект SQLConnection применяется практически всеми компонентами GUI JDBC.**

Для работы с SQLConnection необходимо задать свойство URL. Это можно сделать в конструкторе или с помощью метода setURL(). Это свойство указывает, с какой базой данных устанавливается соединение. Помимо этого, могут быть заданы следующие необязательные свойства:

- Метод setProperties() позволяет задать набор свойств соединения JDBC.
- Метод setUserName() позволяет указать имя пользователя, установившего соединение.
- Метод setPassword() позволяет указать пароль для установления соединения.

При создании объекта SQLConnection соединение с базой данных не устанавливается. Соединение устанавливается при вызове метода getConnection(). 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.

После завершения работы с соединением закройте объект SQLConnection методом close(). Это позволит освободить ресурсы JDBC как на клиенте, так и на сервере.

Кнопки и пункты меню:

Объект SQLStatementButton представляет кнопку, при нажатии которой передается на выполнение запрос SQL. Класс SQLStatementButton расширяет класс JButton из комплекта JFC, поэтому данная кнопка выглядит стандартно.

Аналогично, объект SQLStatementMenuItem представляет пункт меню, при выборе которого выполняется запрос SQL. 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().

Класс SQLStatementDocument:

Класс SQLStatementDocument - это реализация интерфейса Document JFC. 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(). Для обработки оператора SQL, указанного в документе, служит метод execute().

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 ();
```

Результаты обработки оператора SQL можно получить методами getResultSet(), getMoreResults(), getUpdateCount() и getWarnings().

Класс SQLResultSetFormPane:

Класс SQLResultSetFormPane представляет форму с результатами обработки запроса SQL. 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. Это можно сделать в конструкторе или с помощью методов setConnection() и setQuery(). Метод load() позволяет передать запрос на обработку и получить первую запись результатов. Для того чтобы закрыть объект с набором результатов обработки запроса, вызовите метод close().

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);
```

Класс SQLResultSetTablePane:

Класс SQLResultSetTablePane представляет результаты запроса SQL в виде таблицы. 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. Это можно сделать в конструкторе или с помощью методов setConnection() и setQuery(). Для передачи запроса на выполнение и загрузки результатов в таблицу вызовите метод load(). Для того чтобы закрыть объект с набором результатов обработки запроса, вызовите метод close().

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

Класс SQLResultSetTableModel:

Объект SQLResultSetTablePane реализован по принципу ″модель-визуализация-управление″, согласно которому данные и пользовательский интерфейс разнесены в разные классы. Такая реализация позволяет интегрировать SQLResultSetTableModel с классом JTable JFC. 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.

Для работы с объектом SQLResultSetTableModel нужно задать свойства connection и query. Это можно сделать в конструкторе или с помощью методов setConnection() и setQuery(). Для передачи запроса на выполнение и загрузки результатов вызовите метод load(). Для того чтобы закрыть объект с набором результатов обработки запроса, вызовите метод close().

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);
```

Редактор запросов SQL:

SQLQueryBuilderPane - это интерактивный инструмент для динамического создания запросов SQL.

Для работы с SQLQueryPane нужно задать свойство connection. Это можно сделать в конструкторе или с помощью метода setConnection(). Метод load() позволяет загрузить данные, необходимые графическому интерфейсу редактора запросов. Метод getQuery() позволяет получить созданный пользователем запрос SQL.

Следующий фрагмент кода создает объект SQLQueryBuilderPane и добавляет его во фрейм:

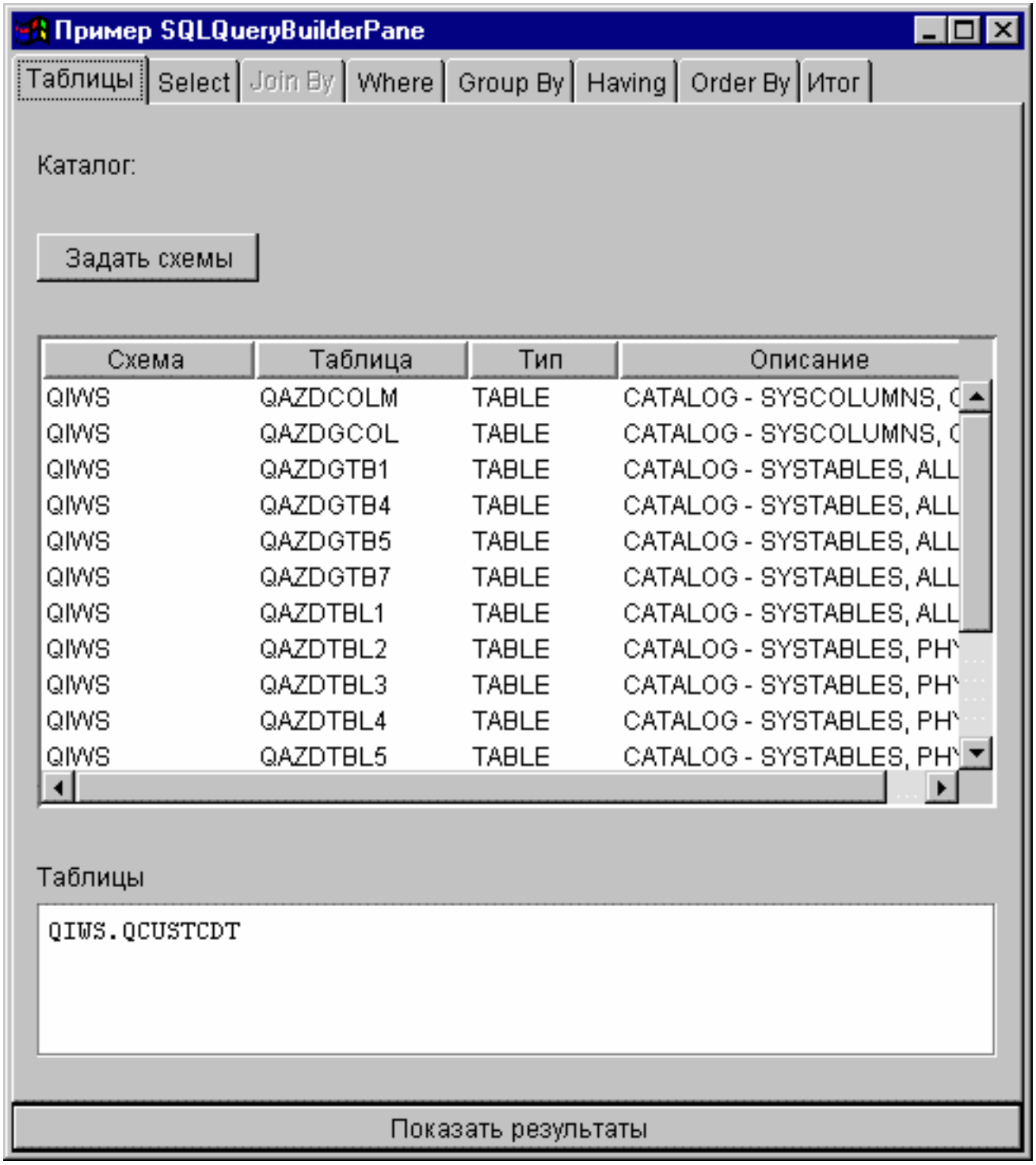
```
                    // Создание объекта SQLQueryBuilderPane.
                    // Предполагается, что объект типа
                    // "connection" - это объект SQLConnection,
                    // который уже создан и инициализирован.
SQLQueryBuilderPane queryBuilder = new SQLQueryBuilderPane (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

Выводится панель SQLQueryBuilderPane и кнопка. 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



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:

- Объект VJobList - это ресурс, представляющий список заданий iSeries. Он рассчитан на применение с объектами AS400Panes.

- Объект VJob - это ресурс, представляющий список сообщений в протоколе задания. Он рассчитан на применение в панелях 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. Это можно сделать в конструкторе или с помощью методов setSystem(), setName(), setNumber() и setUser().

Для применения объекта VJob необходимо задать свойство system. Задайте это свойство с помощью конструктора или метода setSystem().

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. С помощью метода setName() укажите, что будут показаны только задания с определенным именем. С помощью метода setNumber() укажите, что будут показаны только задания с определенным номером. Аналогично, с помощью метода setUser() укажите, что будут показаны только задания определенного пользователя.

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() вы можете ограничить диапазон действий, разрешенных пользователям.

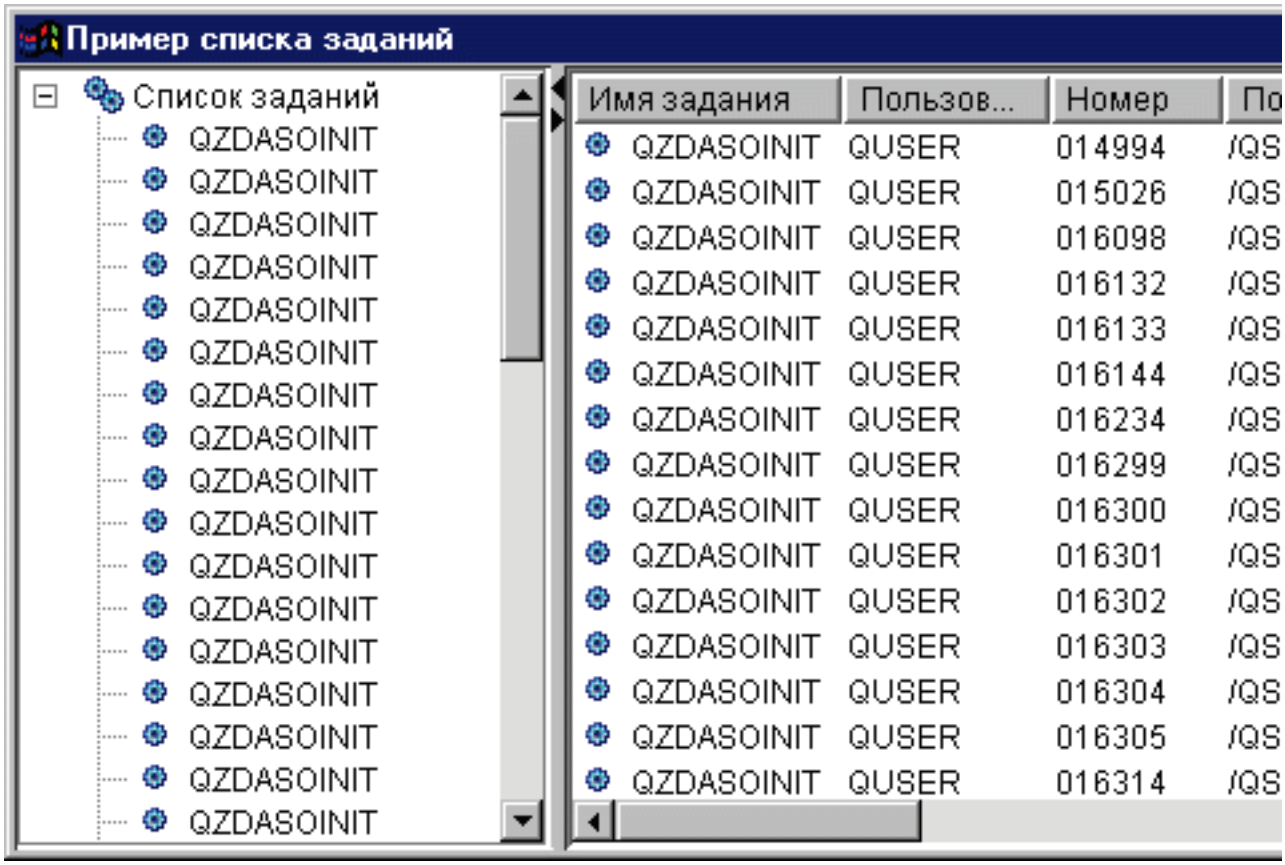
Ниже приведен пример создания объекта VJobList и его представления в окне объекта AS400ExplorerPane:

```
                    // 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:



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:

Объект VMessageList - это ресурс, представляющий список сообщений. Данный ресурс применяется с объектами 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. Это можно сделать в конструкторе или с помощью метода setMessageList(). 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

Класс VMessageQueue:

Объект VMessageQueue - это ресурс, представляющий сообщения из очередей сообщений сервера. Он применяется с объектами AS400Pane.

Для работы с объектом VMessageQueue необходимо задать свойства system и path. Это можно сделать с помощью конструктора 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. Задайте серьезность выводимых сообщений с помощью метода setSeverity(). Задайте тип выводимых сообщений с помощью метода setSelection().

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.

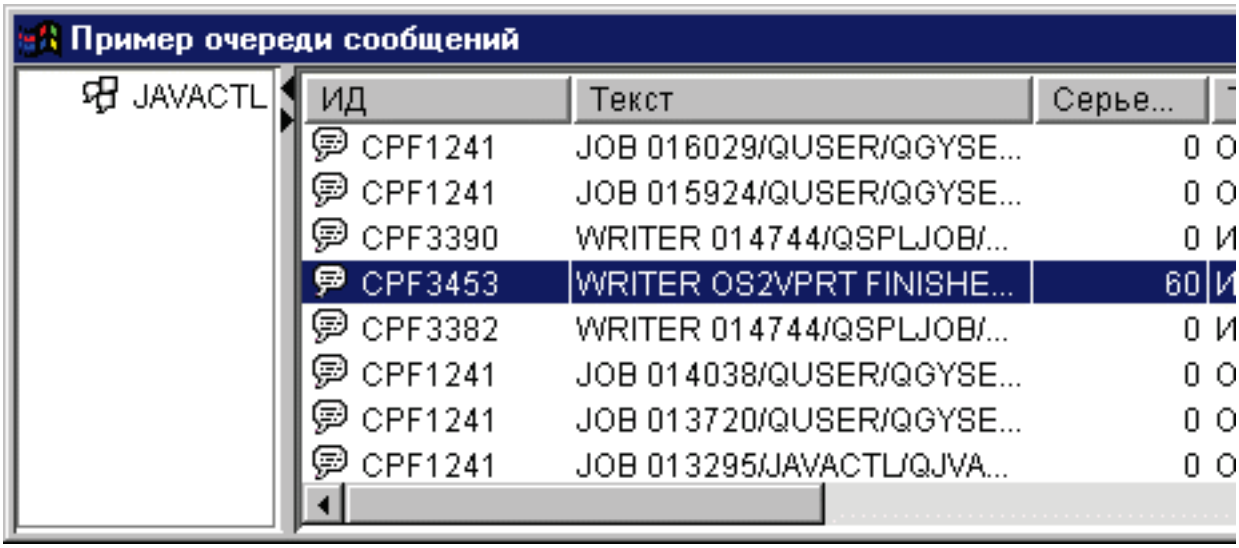
Следующий фрагмент кода создает объект VMessageQueue и выводит его содержимое в панели AS400ExplorerPane:

```
                    // 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



Классы прав доступа

Для получения информации о правах доступа в графическом пользовательском интерфейсе (GUI) применяются классы VIFSFile и VIFSDirectory. 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

С помощью перечисленных ниже компонентов пакета Vaccess программы на Java могут показывать списки ресурсов печати сервера в окнах GUI.

- 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:

Объект VPrinters - это ресурс, представляющий список принтеров. Он рассчитан на применение с объектом AS400Pane.

Для работы с объектом VPrinters необходимо правильно задать свойство system. Задайте это свойство с помощью конструктора или метода setSystem().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. Фильтр для отбора принтеров можно задать с помощью метода setPrinterFilter().

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.

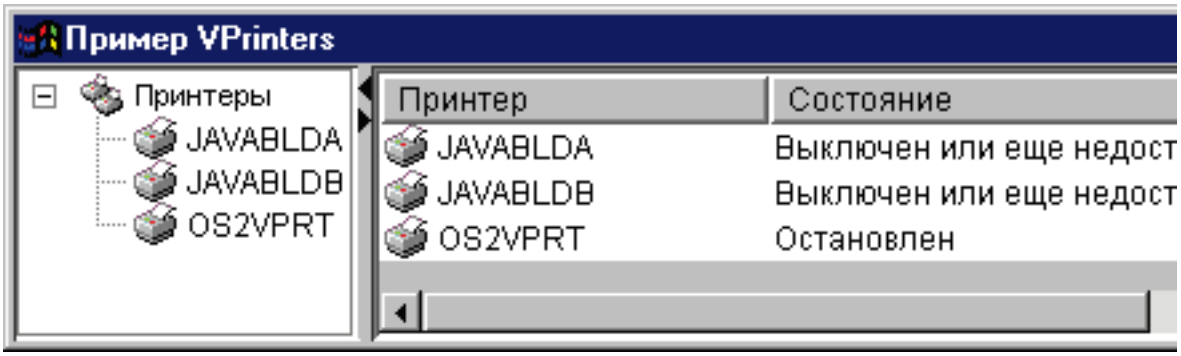
Следующий фрагмент кода создает объект VPrinters и выводит его в панели AS400TreePane:

```
                    // 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



Класс VPrinter:

Объект VPrinter - это ресурс, представляющий принтер сервера и его буферные файлы. Этот объект применяется с объектами AS400Pane.

Для работы с объектом VPrinter необходимо правильно задать свойство printer. Это можно сделать в конструкторе или с помощью метода setPrinter(). 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.

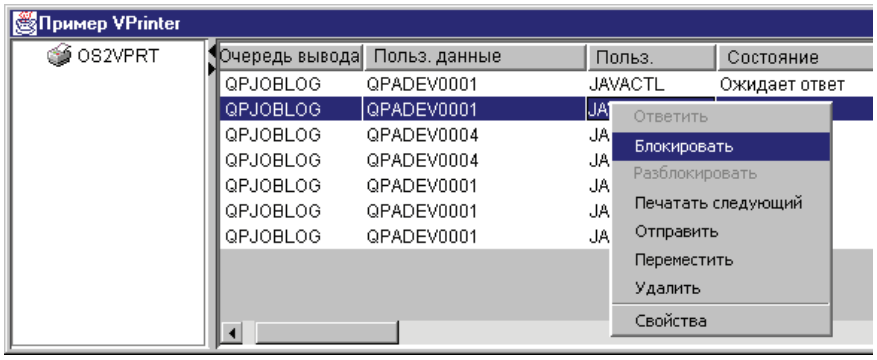
Следующий фрагмент кода создает объект VPrinter и показывает его в панели AS400ExplorerPane:

```
                    // 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);
```

Пример

Показывает список ресурсов печати на панели AS400ExplorerPane с помощью объекта VPrinter. На рис. 1 показан компонент GUI VPrinter:

Рисунок 1: Компонент GUI VPrinter



Класс VPrinterOutput:

Объект VPrinterOutput - это ресурс, представляющий список буферных файлов сервера с помощью объекта AS400Pane.

Для работы с объектом VPrinterOutput необходимо задать свойство system. Это можно сделать с помощью конструктора или метода setSystem(). 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. Метод setFormTypeFilter() позволяет указать типы выводимых форм. Метод setUserDataFilter() позволяет выбрать пользовательские данные для вывода. Метод setUserFilter() позволяет указать пользователей, буферные файлы которых будут показаны.

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.

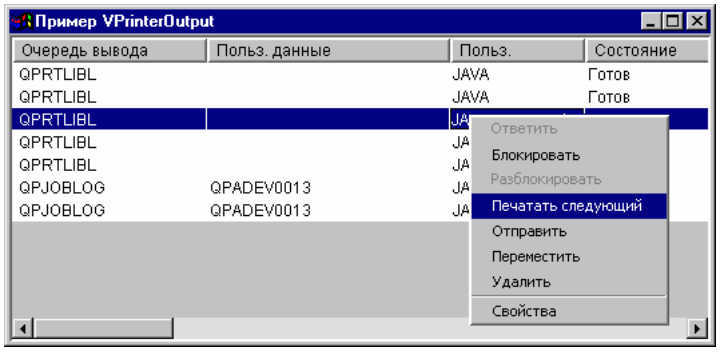
Следующий фрагмент кода создает объект VPrinterOutput и выводит его в панели AS400ListPane:

```
                    // 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);
```

Пример

Показывает список буферных файлов с помощью объекта VPrinterOutput. На рис. 1 показан компонент GUI VPrinterOutput:

Рисунок 1: Компонент GUI VPrinterOutput



Класс SpooledFileViewer:

Класс SpooledFileViewer создает окно для просмотра файлов AFP и SCS, буферизованных для печати. 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. Конструктор SpooledFileViewer() позволяет создать программу просмотра, не указывая связанный с ней буферный файл. В этом случае буферный файл нужно будет задать позже с помощью метода setSpooledFile(SpooledFile). Конструктор SpooledFileViewer(SpooledFile) создает программу просмотра, в окно которой будет загружена первая страница указанного буферного файла. Конструктор SpooledFileViewer(spooledFile, int) создает программу просмотра, в окно которой будет загружена указанная страница заданного буферного файла. После создания программы просмотра любым из описанных способов необходимо вызвать метод load() для загрузки данных буферного файла.

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()

Работа программы должна завершаться вызовом метода close(), который закрывает поток ввода и освобождает все связанные с ним ресурсы.

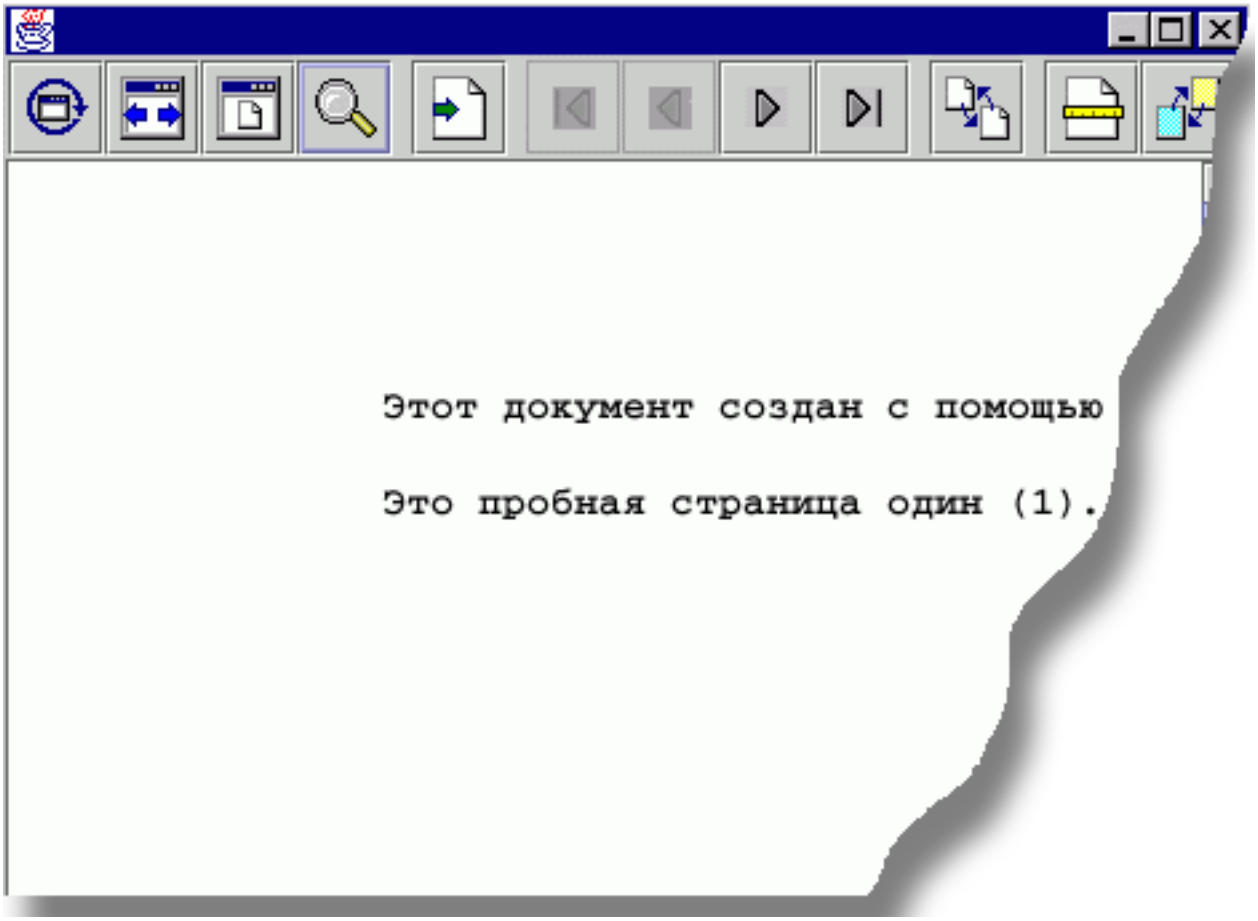
Применение класса SpooledFileViewer

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.

Классы ProgramCall Vaccess

С помощью компонентов Vaccess, предназначенных для вызова программ, программы на Java могут создавать кнопки и меню для вызова программ сервера. Для задания входных параметров, выходных параметров и параметров смешанного типа применяются объекты ProgramParameter. В выходных параметрах и параметрах смешанного типа возвращается результат выполнения программы сервера.

Объект ProgramCallButton представляет кнопку, при нажатии которой запускается программа сервера. Класс ProgramCallButton расширяет класс JButton из комплекта JFC, поэтому данная кнопка выглядит стандартно.

Аналогично, объект ProgramCallMenuItem представляет пункт меню, при выборе которого запускается программа на сервере. 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. Для определения момента запуска программы сервера добавьте к кнопке или пункту меню обработчик ActionCompletedListener с помощью метода addActionCompletedListener(). При запуске программы всем обработчикам будет передано событие ActionCompletedEvent. 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

Для передачи данных программе сервера и получения данных от этой программы в приложении на Java применяются объекты ProgramParameter. Входные данные задаются методом . method. После выполнения программы выходные данные можно получить методом getOutputData.

Каждый параметр представляет собой массив байтов. Приведение данных к нужным форматам 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:

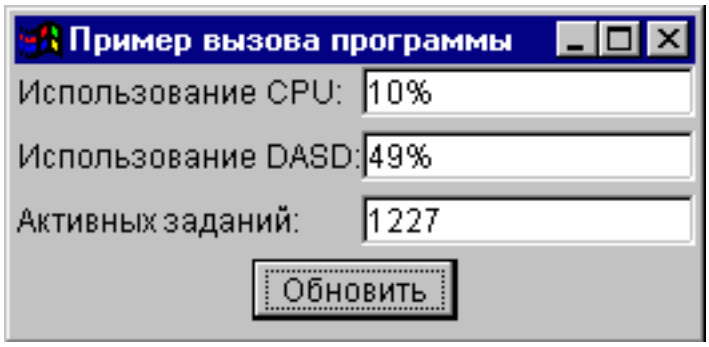
```
                    // Первый параметр - строка длиной
                    // до 100 символов.
                    // Это входной параметр. Предполагается,
```

```
                    // 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



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, 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:

Класс RecordListFormPane представляет в форме содержимое файла сервера. 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. Это можно сделать с помощью конструктора или методов setSystem() и setFileName(). Метод load() загружает содержимое файла и выводит первую запись. После завершения работы с содержимым файла вызовите метод close(), чтобы закрыть файл.

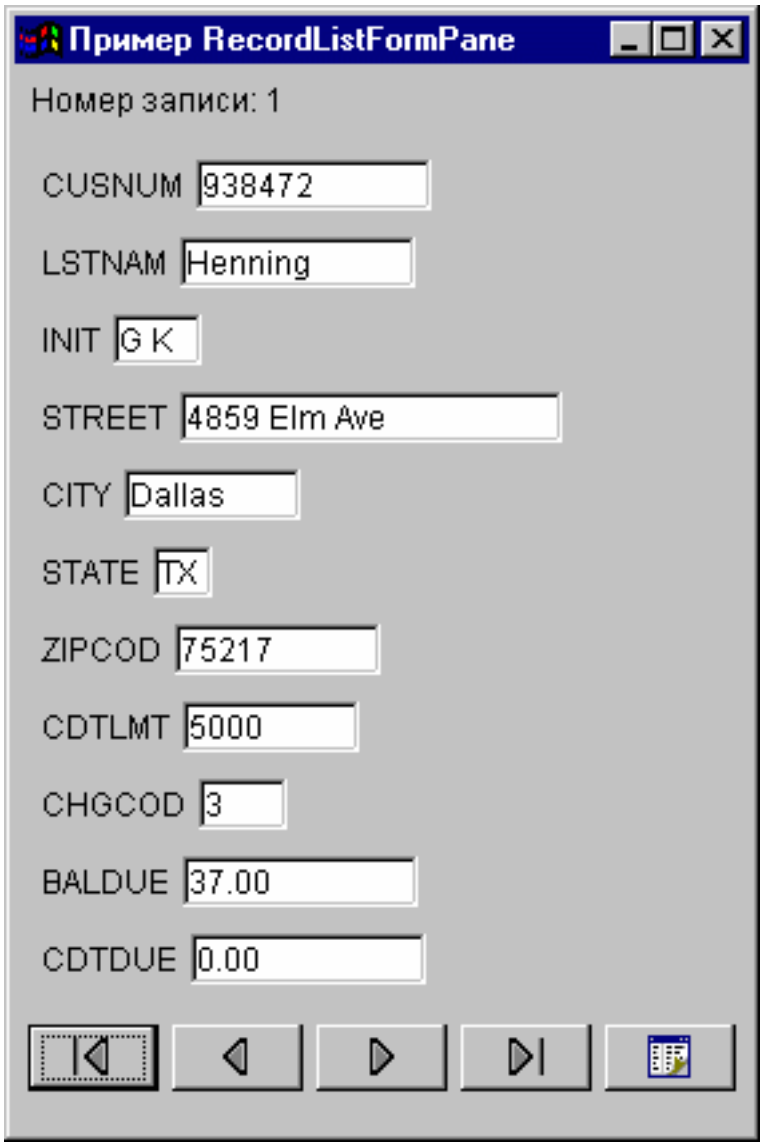
Следующий фрагмент кода создает объект RecordListFormPane и добавляет его во фрейм:

```
                    // 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



Класс RecordListTablePane:

Класс RecordListTablePane представляет содержимое файла сервера в виде таблицы. Каждая строка таблицы соответствует одной записи файла, а каждый столбец строки - одному полю.

Для работы с объектом RecordListTablePane необходимо задать свойства system и fileName. Это можно сделать с помощью конструктора или методов setSystem() и setFileName(). Метод load() загружает содержимое файла в таблицу. После завершения работы с содержимым файла вызовите метод close(), чтобы закрыть файл.

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);
```

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. Такая реализация позволяет интегрировать класс RecordListTableModel с классом JTable JFC. 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. Это можно сделать с помощью конструктора или методов setSystem() и setFileName(). Метод load() загружает содержимое файла. После завершения работы с содержимым файла вызовите метод close(), чтобы закрыть файл.

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);
```

Классы ResourceListPane и ResourceListDetailsPane

Классы ResourceListPane и ResourceListDetailsPane предназначены для вывода списка ресурсов средствами 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:

```
// Создание списка ресурсов.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RUserList userList = new RUserList(system);

// Создание панели ResourceListDetailsPane.
// В этом примере таблица содержит два столбца.
// Первый столбец содержит значки и имена пользователей.
// Второй столбец содержит описание, связанное с
// пользователем.
Object[] columnAttributeIDs = new Object[] { null, RUser.TEXT_DESCRIPTION };
ResourceListDetailsPane detailsPane = new ResourceListDetailsPane();
detailsPane.setResourceList(userList);
detailsPane.setColumnAttributeIDs(columnAttributeIDs);

// Добавление объекта ResourceListDetailsPane во фрейм JFrame и вывод фрейма.
JFrame frame = new JFrame("My Window");
frame.getContentPane().add(detailsPane);
frame.pack();
frame.show();

// Необходимо загрузить данные в объект ResourceListDetailsPane.
// Данные из системы iSeries можно получить
// в любой момент.
detailsPane.load ();
```

Классы состояния системы

Компоненты пакета Vaccess, предназначенные для работы с состоянием системы, позволяют создавать объекты GUI на базе объектов AS400Pane. Помимо этого, можно создавать GUI с помощью Java Foundation Classes (JFC). Объект VSystemStatus представляет состояние сервера. Объект VSystemPool представляет системный пул сервера. Объект VSystemStatusPane - это ресурс, представляющий панель с информацией о состоянии системы.

Класс VSystemStatus позволяет получать сведения о состоянии сеанса сервера в среде GUI.

- Метод getSystem() выдает имя сервера, с которым работает класс.
- Метод getText() выдает описание системы
- Метод setSystem() задает сервер, с которым будет работать класс

Помимо вышеуказанных методов, предусмотрены методы просмотра и изменения информации в системном пуле с помощью GUI.

Класс VSystemStatus применяется совместно с панелью VSystemStatusPane. Панель VSystemPane может использоваться для работы не только с состоянием системы, но и с информацией системного пула.

Класс VSystemPool:

С помощью класса VSystemPool можно просматривать и изменять информацию о системных пулах сервера с помощью GUI. Класс VSystemPool применяет различные панели из пакета vaccess, в том числе панель VSystemStatusPane.

Ниже перечислены некоторые методы класса VSystemPool:

- Метод getActions() возвращает список действий, которые разрешено выполнять
- Метод getSystem() возвращает имя сервера, на котором была найдена информация о системном пуле
- Метод setSystemPool() задает системный пул, с которым пользователь планирует работать

Класс VSystemStatusPane:

Класс VSystemStatusPane позволяет программе на Java показывать состояние системы и информацию о системных пулах.

В класс VSystemStatusPane входят следующие методы:
- getVSystemStatus(): Показывает информацию VSystemStatus на панели VSystemStatusPane.
- setAllowModifyAllPools(): Разрешает или запрещает изменение информации о системных пулах.

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();
```

Системные значения

Компоненты пакета Vaccess, предназначенные для работы с системными значениями, позволяют программам на Java создавать GUI на базе существующих объектов AS400Pane, а также создавать собственные панели с помощью классов JFC. Объект VSystemValueList представляет список системных значений сервера.

Для работы с компонентом GUI, представляющим системные значения, необходимо задать имя системы с помощью конструктора или метода setSystem().

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();
```

Классы Vaccess для работы с пользователями и группами

Компоненты пакета Vaccess, предназначенные для работы с пользователями и группами, позволяют создавать списки пользователей и групп на базе класса VUser.

Предусмотрены следующие компоненты:
- Объекты AS400Pane - это компоненты GUI, предназначенные для работы с ресурсами сервера.
- Объект VUserList - это ресурс, представляющий список пользователей и групп сервера. Он рассчитан на применение с объектами AS400Pane.
- Объект VUserAndGroup - это ресурс, представляющий группу пользователей сервера. Он рассчитан на применение с объектами AS400Pane. 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. Это можно сделать с помощью конструктора или метода setSystem(). 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.

- Метод setUserInfo() позволяет задать типы пользователей, которые должны быть показаны в списке.
- Метод setGroupInfo() позволяет указать имя группы.

Объект VUserAndGroup позволяет получить информацию о пользователях и группах системы. Перед получением информации ее необходимо загрузить из системы с помощью метода load. Имя сервера, из которого была загружена информация, можно узнать с помощью метода getSystem.

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() вы можете 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:

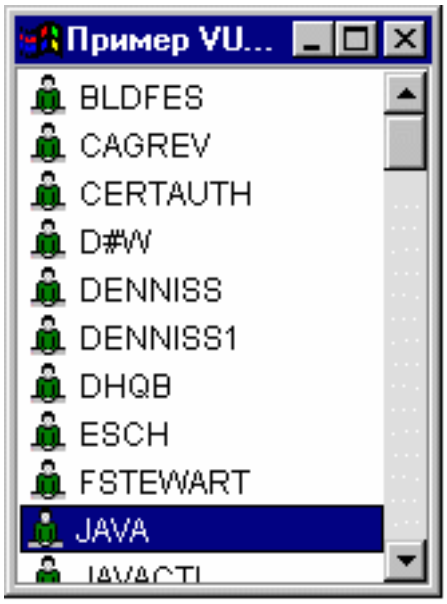
```
// Создать объект VUserAndGroup.
// Предполагается, что объект AS400 уже создан и инициализирован.
VUserAndGroup root = new VUserAndGroup(system);

// 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:



Graphical Toolbox

Graphical Toolbox - это набор инструментов, позволяющий создавать панели пользовательского интерфейса для приложений на Java, аплетов и встраиваемых модулей Навигатора iSeries. В панели можно разместить данные, полученные из системы iSeries или из другого источника, например, локального файла или удаленной программы.

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 (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 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.

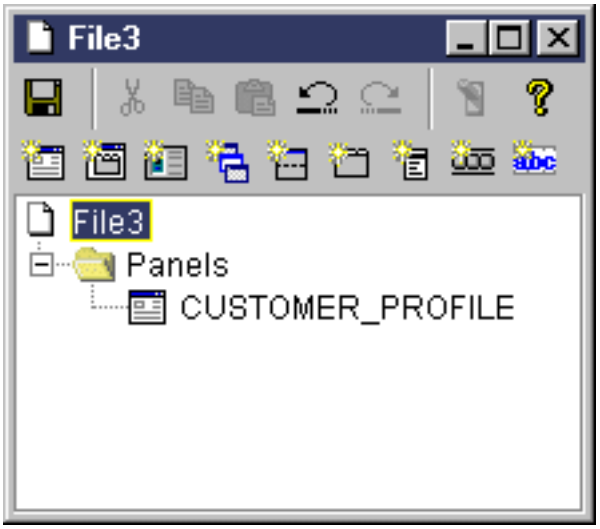
GUI Builder

При запуске программы GUI Builder появляются два окна, показанные на рис. 1:

Рис. 1: Окна GUI Builder

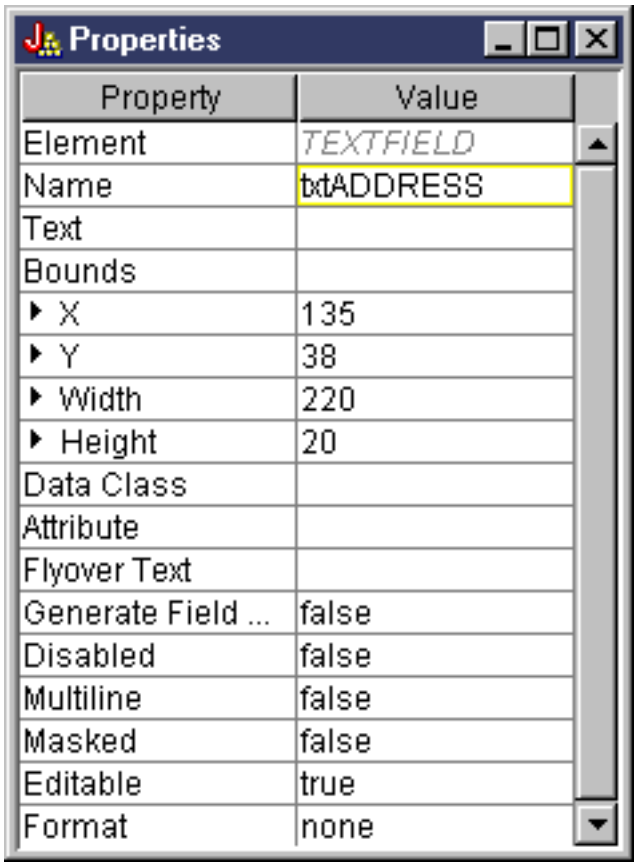
Создание и редактирование файлов PDML с помощью окна Редактор файлов.

Рис. 2: Окно Редактор файлов



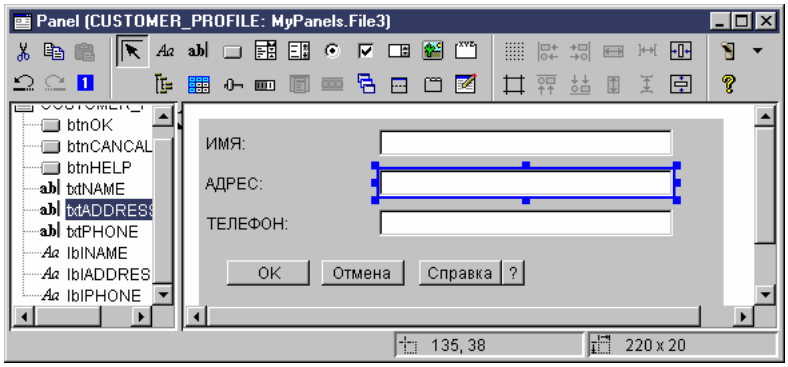
Окно Свойства предназначено для просмотра и изменения свойств выделенного управляющего элемента.

Рис. 3: Окно Свойства



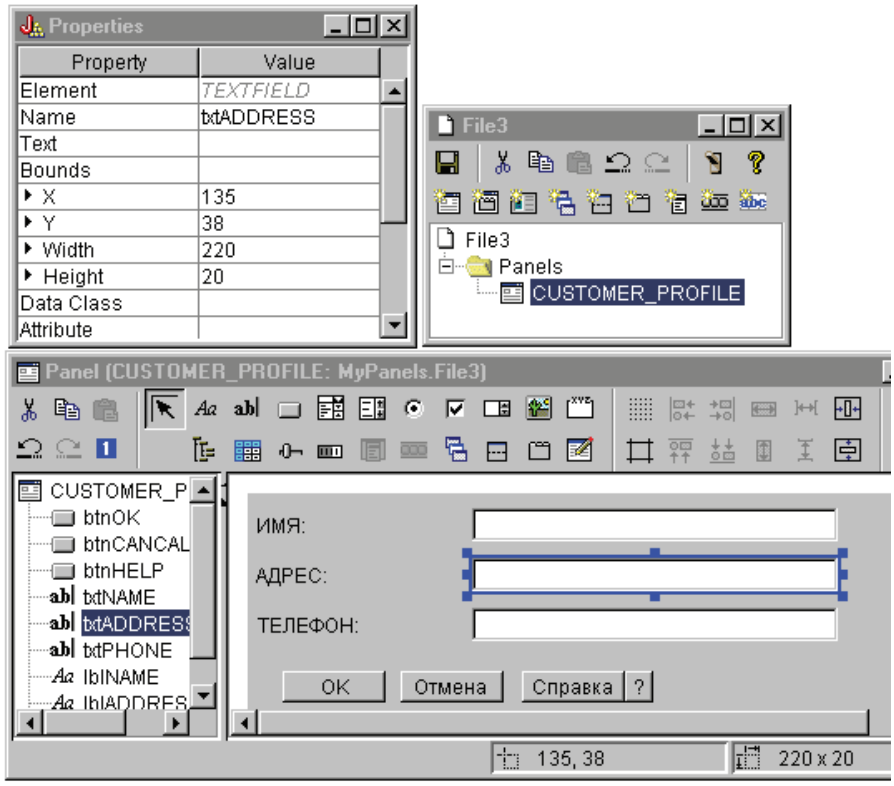
Создание и редактирование компонентов GUI с помощью окна Редактор панелей. Выберите компонент на панели и щелкните мышью в той точке панели, где вы хотите его разместить. На панели инструментов расположены значки для выравнивания группы элементов, предварительного просмотра панели и вызова электронной 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:

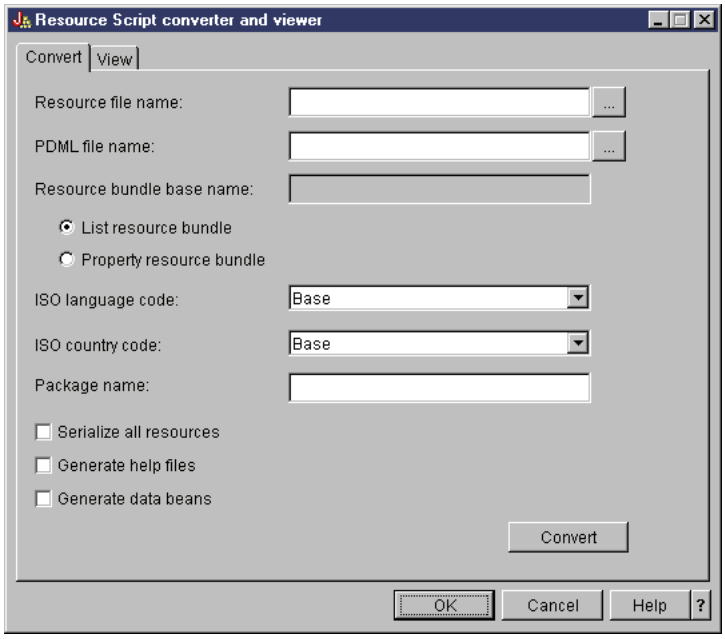
Рис. 5: Группа окон GUI Builder



Resource Script Converter

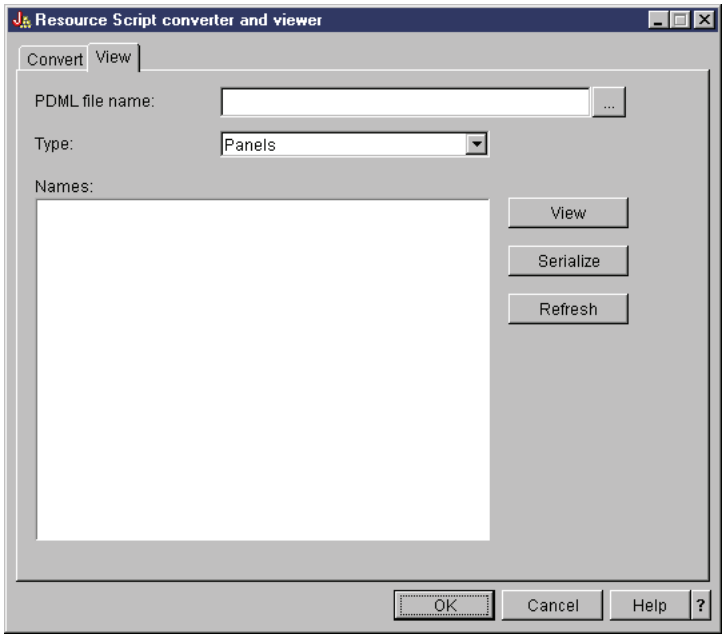
Окно программы Resource Script Converter - это окно диалога с закладками, состоящее из двух панелей. На панели **Преобразовать** вы должны указать имя файла .rc в формате Microsoft или VisualAge for Windows, который необходимо преобразовать в формат PDML. Кроме того, вы можете задать имя целевого файла 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



Дополнительная информация о Graphical Toolbox

Дополнительная информация о Graphical Toolbox приведена в следующих разделах:

- Настройка Graphical Toolbox
- Создание пользовательского интерфейса

- Динамический просмотр панелей
- Создание файлов с электронной справкой
- Пример работы с Graphical Toolbox
- Работа с Graphical Toolbox в браузере
- Панель инструментов Редактора панелей

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

Если при создании приложений на Java вы планируете применять Graphical Toolbox, то установите его файлы .jar на своей рабочей станции. Это можно сделать двумя способами:

Передача файлов .jar

Примечание: Ниже описано несколько различных способов передачи файлов .jar. Перед тем как начать передачу файлов, убедитесь, что в системе iSeries установлена лицензионная программа IBM Toolbox for Java. 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.
- Подключите сетевой диск с помощью iSeries Access для Windows.
- Установите файлы .jar набора Graphical Toolbox с помощью класса AS400ToolboxInstaller, входящего в набор классов IBM Toolbox for Java. Для этого укажите имя пакета ″OPNAV″. За дополнительной информацией обратитесь к разделу Классы установки и обновления клиента.

Установка файлов .jar с помощью iSeries Access для Windows

Вы можете установить Graphical Toolbox во время установки iSeries Access для Windows. IBM Toolbox for Java теперь поставляется вместе с iSeries Access для Windows. Если программа iSeries Access для Windows не установлена, выберите опцию настраиваемой установки, а затем в меню установки выберите компонент **IBM Toolbox for Java**. Если программа iSeries Access для Windows уже установлена, укажите опцию выборочной установки, а затем выберите компонент IBM Toolbox for Java, если он еще не установлен.

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. 412

| Если пакет Graphical Toolbox был установлен вместе с iSeries Access для Windows, то файлы .jar (кроме файла jhall.jar) будут расположены в каталоге **\Program Files\Ibm\ClientAccess\jt400\lib** установочного диска iSeries Access для Windows. Программа iSeries Access для Windows устанавливает файл jhall.jar в каталоге **\Program Files\Ibm\Client Access\jre\lib**. Соответствующие пути к файлам должны быть указаны в переменной CLASSPATH.

Описание файлов .jar

- **uitools.jar**: Содержит инструменты GUI Builder и Resource Script Converter.
- **jui400.jar**: Содержит динамические API Graphical Toolbox. Программы на Java применяют этот API для вывода панелей, созданных с помощью инструментов. Эти классы могут распространяться вместе с приложениями.
- **data400.jar**: Содержит динамический API языка PCML. Этот API может применяться в программах на Java для вызова программ iSeries, параметры и возвращаемые значения которых описаны на PCML. Эти классы могут распространяться вместе с приложениями.
- **util400.jar**: Содержит классы утилит для форматирования данных и обработки сообщений iSeries. Эти классы могут распространяться вместе с приложениями.
- **jhall.jar**: Содержит классы JavaHelp, которые позволяют просматривать электронную справку и контекстную справку для панелей, созданных с помощью GUI Builder.
| • **Анализатор XML**: Содержит анализатор XML, с помощью которого классы API интерпретируют
| документы PDML и PCML.

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. Соответствующие файлы JAR расположены на сервере iSeries в каталоге **/QIBM/ProdData/HTTP/Public/jt400/Mri29xx**, где 29xx - 4-разрядный код NLV OS/400, соответствующий вашему языку, а также стране или региону. (К именам файлов JAR в каталогах Mri29xx добавлен двухсимвольный суффикс, содержащий код языка и код страны или региона Java.) 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

Для запуска GUI Builder введите следующую команду:

```
java com.ibm.as400.ui.tools.GUI Builder [-plaf вид интерфейса]
```

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.

Опции `-plaf` *вид интерфейса*

Задает внешний вид интерфейса на данной платформе. Эта опция позволяет переопределить значение по умолчанию, которое устанавливается в зависимости от текущей платформы. С ее помощью вы можете посмотреть, как панель будет выглядеть в различных операционных системах. Возможны следующие значения:

- 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. В строке меню окна GUI Builder выберите **Файл --> Создать файл**. После создания файла PDML вы можете определить для него любые из следующих типов ресурсов пользовательского интерфейса.

Панель
Основной тип ресурса. Он описывает прямоугольную область экрана, в которой расположены элементы пользовательского интерфейса. К ним относятся простые управляющие элементы, например, радиокнопки и текстовые поля, изображения, анимация, пользовательские управляющие элементы и более сложные вложенные панели (см. приведенное ниже описание разделенной панели, составной панели и панели с закладками). Панель определяет простое окно или окно диалога, либо панель, расположенную в другом ресурсе пользовательского интерфейса.

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
Отдельное окно или окно диалога, состоящее из панелей с закладками, на которых расположены кнопки OK, Отмена и Справка. Описания окон с закладками хранятся в ресурсе панели.

Мастер
Отдельное окно или окно диалога, представляющее набор панелей, которые выводятся на экран в предопределенной последовательности. На панелях могут быть расположены кнопки Назад, Вперед, Отмена, Готово и Справка. Кроме того, в левой области окна мастера может быть показан список задач, в котором задачи выделяются по мере их выполнения.

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. Необходимо добавить содержимое справки в соответствующих точках файла HTML в зоне действия тегов <!-- `HELPDOC:SEGMENTBEGIN` --> и <!-- `HELPDOC:SEGMENTEND` --> tags. Более подробная информация о справке приведена в разделе Редактирование документов справки, созданных GUI builder.

Для компонентов JavaBean, предоставляющих данные для панели, можно создать шаблон исходного кода. В окне Свойства программы GUI Builder укажите свойства DATACLASS и ATTRIBUTE для тех управляющих элементов, которые будут содержать данные. Свойство DATACLASS задает имя класса компонента, а свойство ATTRIBUTE - имя метода gettor или settor, который реализован в классе компонента. После сохранения этой информации в файле PDML создайте шаблоны исходного кода Java с помощью GUI Builder и откомпилируйте их. После запуска программы данные будут выведены на панель с помощью указанных методов gettor и settor.

Примечание: Количество и тип методов gettor и settor зависит от типа управляющего элемента пользовательского интерфейса, с которым связаны эти методы. Документация по протоколам методов для каждого управляющего элемента приведена в описании класса DataBean.

Кроме того, вы можете сохранить файл PDML в виде потока байтов. При этом будет создано компактное двоичное представление всех элементов пользовательского интерфейса, расположенных в данном файле. Это позволит значительно повысить производительность пользовательского интерфейса, поскольку файл PDML не будет интерпретироваться при каждом выводе панелей.

Общая информация: Если вы создали файл PDML **MyPanels.pdml**, в зависимости от выбранных опций инструментов будут также автоматически созданы следующие файлы:

- **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
- **<имя-панели>.html** для каждой панели в файле PDML, если вы выбрали создание электронной структуры справки
- **<имя-класса-данных>.java** и **<имя-класса-данных>.class** для каждого отдельного класса компонента, заданного в свойствах DATACLASS, если вы выбрали создание структуры исходного кода для объектов JavaBeans
- **<имя-ресурса>.pdml.ser** для каждого ресурса пользовательского интерфейса, определенного в файле PDML, если для PDML было выбрано сохранение в двоичном файле.

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
```

Если вы не внесли в переменную среды CLASSPATH файлы JAR, Graphical Toolbox, их необходимо указать в командной строке с помощью опции `classpath`. Дополнительная информация приведена в разделе Настройка 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 Имя созданного файла PDML. По умолчанию совпадает с именем обрабатываемого файла RC.

-p имя Имя созданного файла PROPERTIES. По умолчанию совпадает с именем файла PDML.

-r имя Имя созданного подкласса ListResourceBundle. По умолчанию совпадает с именем файла PDML.

-package имя
Имя пакета, с которым будут связаны созданные ресурсы. Если это имя не указано, операторы пакетов созданы не будут.

-l локаль
Локаль создаваемых ресурсов. Если она указана, к имени созданного комплекта ресурсов добавляется соответствующие двухсимвольные коды языка ISO и страны или региона.

-h Создать шаблон электронной справки в формате HTML.

-d Создать шаблон исходного кода компонентов JavaBean.

-s Сохранить все ресурсы в двоичном виде.

Преобразование ресурсов окон в формат PDML

Все окна диалога, меню и таблицы строк, описанные в файле .rc, будут преобразованы в соответствующие ресурсы Graphical Toolbox и сохранены в файле PDML. В новом файле PDML вы можете задать свойства DATACLASS и ATTRIBUTE для управляющих элементов Windows в соответствии с обычными правилами присвоения имен ресурсам Windows. Во время преобразования эти свойства будут использованы при создании шаблонов исходного кода компонентов JavaBean.

Идентификаторы ресурсов Windows задаются в следующем формате:

```
IDCB_<имя-класса>_<атрибут>
```

где `<имя-класса>` - это полное имя класса компонента, который должен быть задан в свойстве DATACLASS управляющего элемента, а `<атрибут>` - это имя свойства компонента, которое должно быть задано в свойстве ATTRIBUTE управляющего элемента.

Например, текстовое поле Windows с ИД ресурса `IDCB_com_MyCompany_MyPackage_MyBean_SampleAttribute` будет преобразовано в свойство DATACLASS **com.MyCompany.MyPackage.MyBean** и свойство ATTRIBUTE класса **SampleAttribute**. Если при преобразовании создаются объекты JavaBean, создается также исходный файл Java **MyBean.java**, содержащий оператор пакета **package com.MyCompany.MyPackage** и методы gettor и settor для свойства **SampleAttribute**.

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.

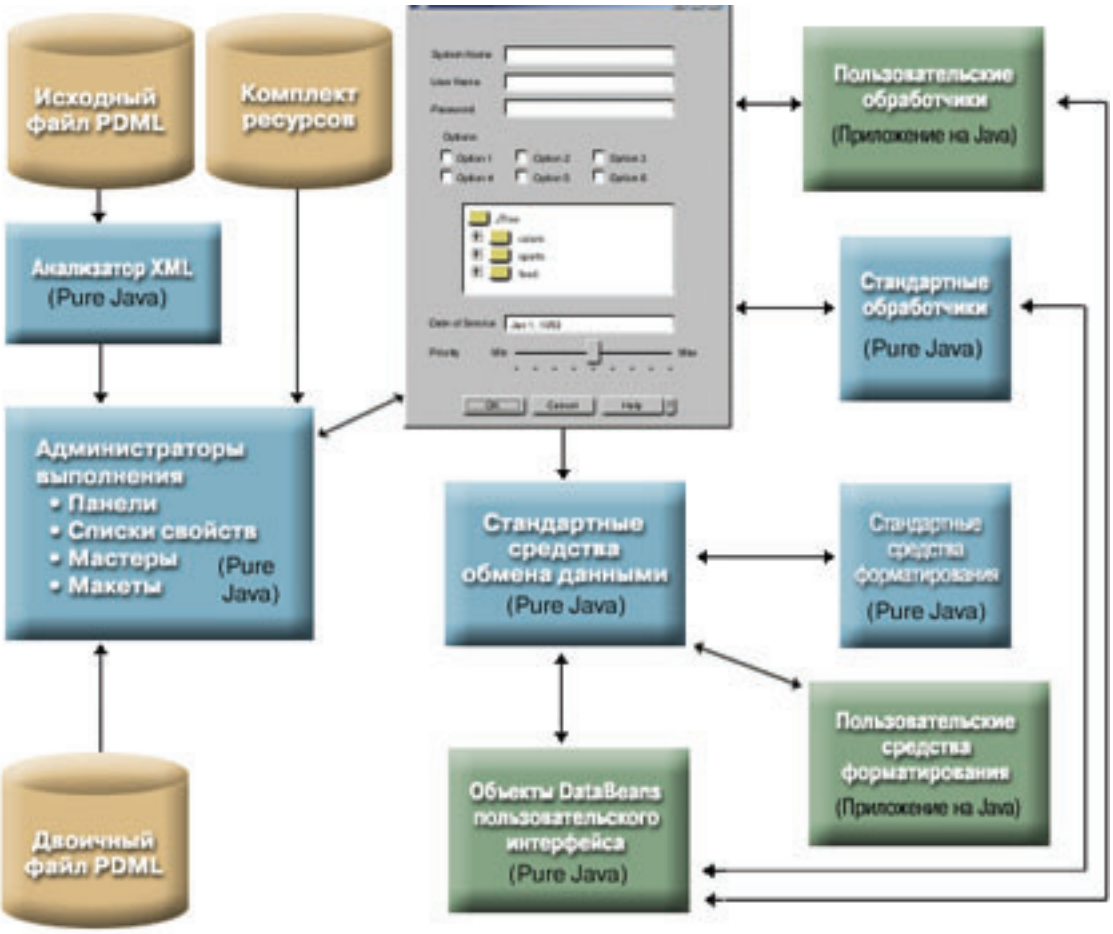
Среда выполнения Graphical Toolbox выполняет следующие задачи:

- Обрабатывает все данные, которыми обмениваются элементы управления пользовательского интерфейса и компоненты JavaBean, 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.)

Динамический API Graphical Toolbox находится в пакете com.ibm.as400.ui.framework.java.

Элементы среды выполнения Graphical Toolbox показаны на Рис. 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.

Подробное описание рисунка 1: Среда выполнения Graphical Toolbox (rzahh504.gif)

IBM Toolbox for Java: Просмотр панелей во время выполнения программы

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)
- Путем вызова Анализатора XML IBM iSeries (голубой прямоугольник в области 4), который обрабатывает исходные файлы PDML (один из двух коричневых цилиндров в области 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>Мой управляющий элемент</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 -->
```

Стандартный нижний колонтитул - </BODY></HTML>Он добавляется в каждый файл HTML.

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:

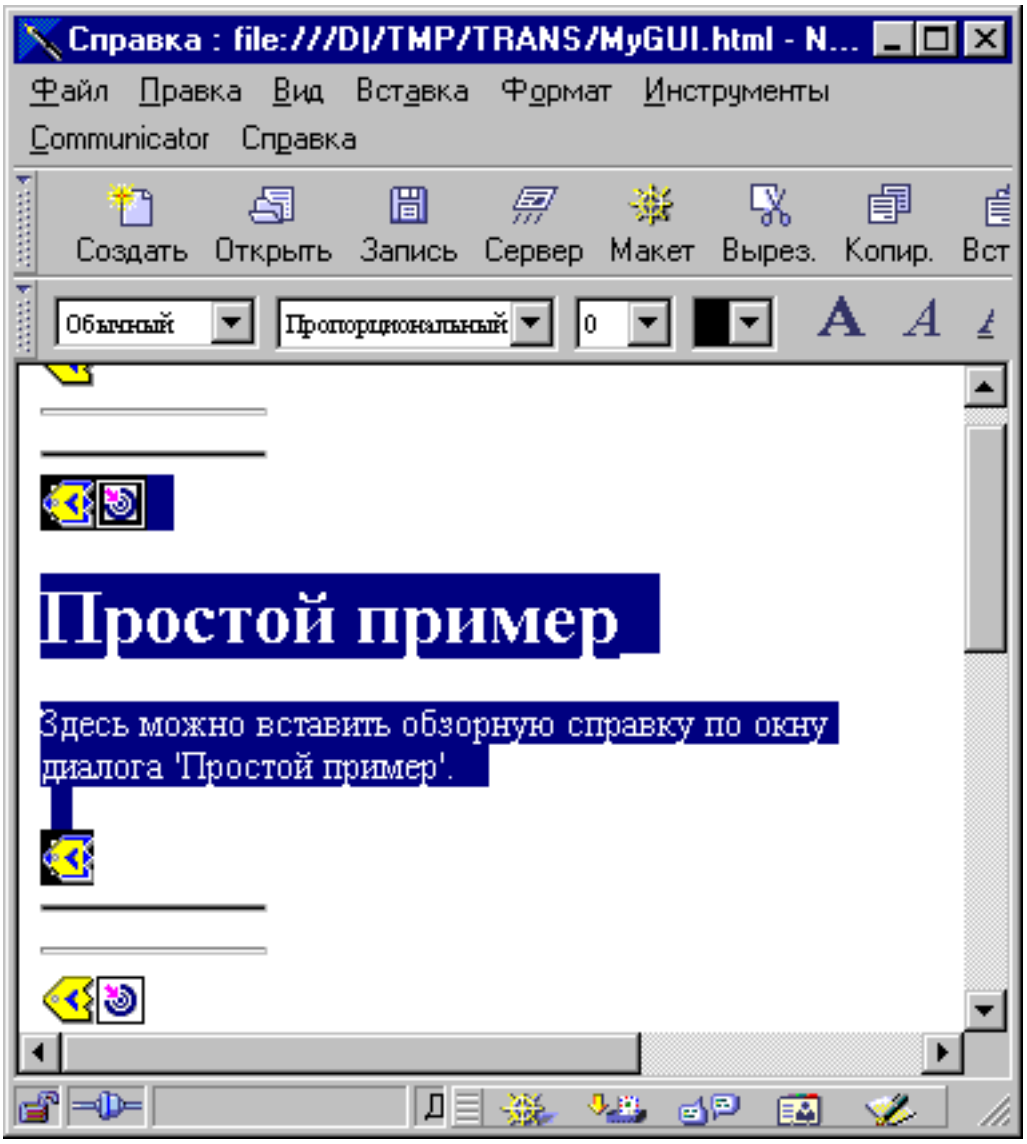
- Внешние URL используются стандартным образом. В них можно указывать внутренние составляющие ссылки.
- Ссылки внутри раздела являются стандартными, но не должны содержать расширения .html в имени тега. Это связано с тем, что программа обработки документа справки считает такие ссылки внешними, если разделы 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>Пример Graphical Toolbox</title>
</head>

<body>
<h1>Пример применения встраиваемого модуля Java(TM) с Graphical Toolbox</h1>
<p>

<!-- НАЧАЛО ТЕГОВ АПЛЕТА ВСТРАИВАЕМОГО МОДУЛЯ JAVA(TM) -->

<!-- В приведенных ниже тегах применяется специальный синтаксис, который -->
<!-- позволяет браузерам Netscape и Internet Explorer загружать -->
<!-- встраиваемый модуль Java и запускать аплеты в среде JRE этого модуля. -->
<!-- Не изменяйте эти теги.           -->
<!-- Дополнительная информация приведена на странице -->
<!-- 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>

<!-- КОНЕЦ ТЕГОВ АПЛЕТА ВСТРАИВАЕМОГО МОДУЛЯ JAVA(TM) -->

<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. 412. 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

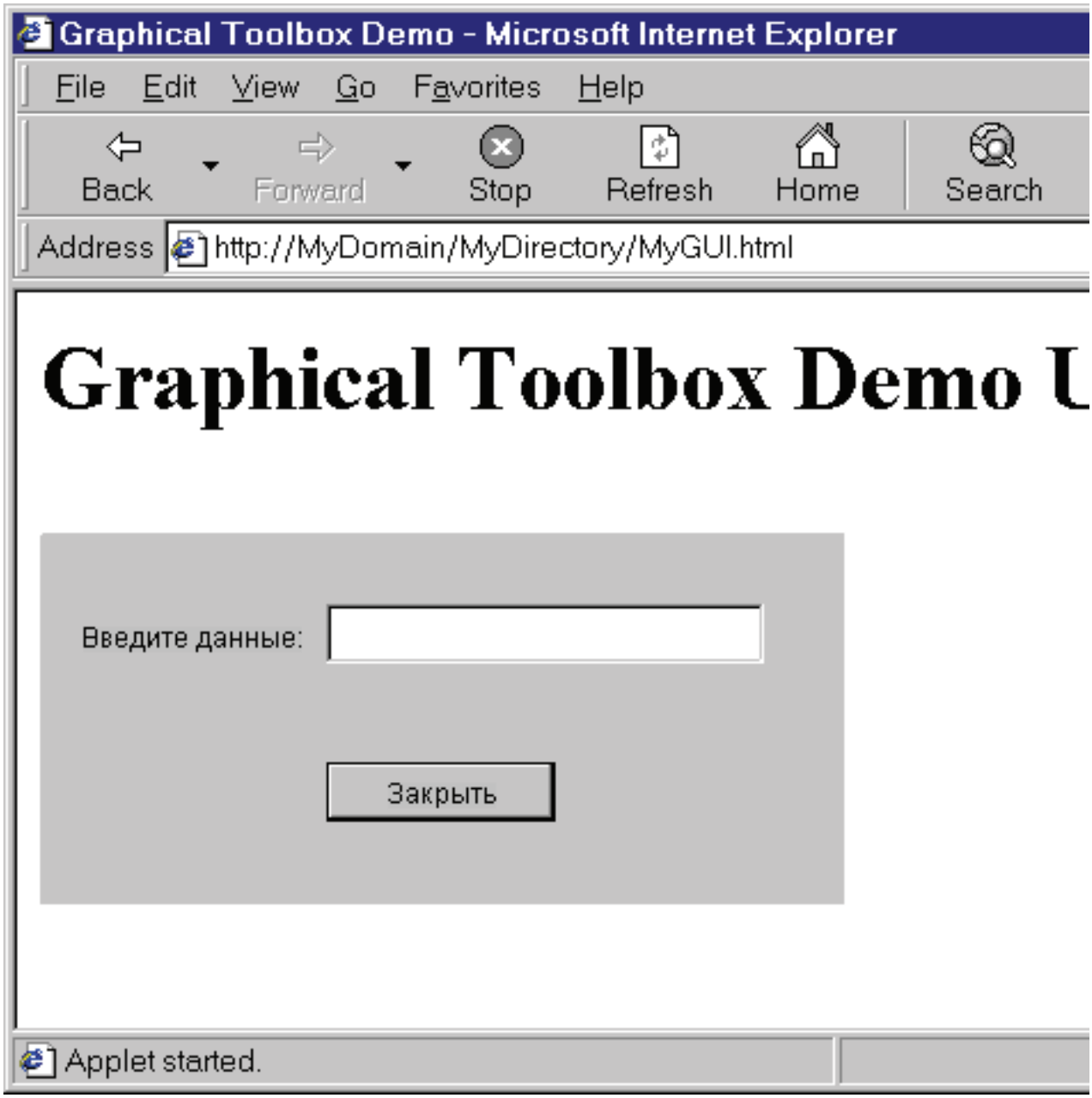
Установите аплет на выбранном Web-сервере, выполнив следующие действия:

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:

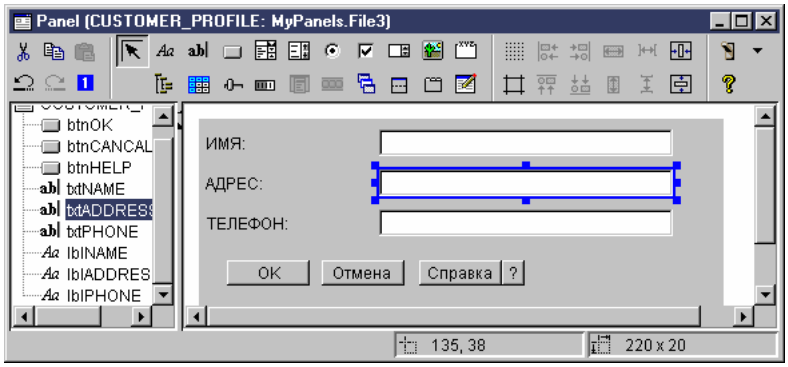
Рис. 1: Запуск примера аплета в браузере



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.

IBM Toolbox for Java - Компоненты JavaBean

Объекты JavaBean(TM) 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.

Компоненты JavaBean могут быть визуальными или невизуальными. Невизуальные компоненты JavaBean всегда имеют визуальное представление (значок или имя), позволяющее работать с ними в визуальных средах.

Все общие классы IBM Toolbox for Java также являются компонентами JavaBean. Эти классы соответствуют стандартам JavaBean компании Javasoft и могут использоваться многократно. Свойства и методы компонента JavaBean IBM Toolbox for Java совпадают со свойствами и методами класса.

Объекты JavaBean могут использоваться в прикладной программе и в средствах визуального программирования, таких как IBM VisualAge для Java.

Примеры

Ниже приведен пример применения JavaBean в программе и создания программы на основе JavaBean с помощью визуального компоновщика:

“Пример: Код компонента IBM Toolbox for Java” на стр. 533

“Пример: Создание объектов JavaBean с помощью визуального компоновщика” на стр. 534

JDBC

JDBC(TM) - 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. Можно также воспользоваться драйвером JDBC IBM Developer Kit for Java, который также называют ’внутренним’ драйвером JDBC:

- Драйвер JDBC IBM Toolbox for Java рекомендуется применять в том случае, если программа на Java и файлы базы данных расположены в разных системах, как, например, в среде клиент-сервер
- Стандартный драйвер JDBC рекомендуется применять в том случае, если программа на Java и файлы базы данных расположены в одной системе iSeries

Дополнительная информация о JDBC и поддержке JDBC в IBM Toolbox for Java, включая список планируемых улучшений, свойства JDBC и не поддерживаемые типы SQL, приведена на следующей странице:

“Классы JDBC” на стр. 63

“Изменения, внесенные в поддержку JDBC продукта IBM Toolbox for Java”

“IBM Toolbox for Java - Свойства JDBC” на стр. 327

“Типы SQL JDBC” на стр. 342

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.
- API JDBC 3.0 (пакеты java.sql и javax.sql) входят в состав J2SE версии 1.4.

Изменения, внесенные в поддержку JDBC продукта IBM Toolbox for Java

В операционной системе OS/400 версии 5, выпуска 3 добавлены следующие функции JDBC:

- 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

Информация о расширенных функциях JDBC для предыдущих выпусков системы приведена в разделе Новые функции поддержки JDBC в IBM Toolbox for Java JDBC для системы V5R2.

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:

Новые возможности JDBC IBM Toolbox for Java позволяют работать с большими объектами (LOB) размером до 2 Гб.

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 OS/400 версии 5, выпуска 2

В операционной системе OS/400 версии 5, выпуска 2 добавлены следующие функции JDBC:

- 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 OS/400 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

В Toolbox for Java теперь применяются те же правила усечения данных, что и в драйвере JDBC продукта IBM Developer Kit for Java. 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:

```
Statement s =
   statement.executeQuery("INSERT INTO MYSCHOOL/MYSTUDENTS (FIRSTNAME) VALUES ('JOHN'");
ResultSet rs = s.getGeneratedKeys();
     // На данный момент сервер iSeries позволяет получить только один
     // ключ -- ключ последней вставленной строки.
rs.next ();
String autoGeneratedKey = rs.getString(1);
     // Автоматически созданный ключ может быть задан, например,
     // 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 OS/400 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 создает 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. Уровень блокировки, заданный в свойстве JDBC Уровень блокировки курсора (если не указаны первые два значения)

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 OS/400 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 создавал исключительную ситуацию при изменении значения на *NONE после создания соединения.

IBM Toolbox for Java - Свойства JDBC

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.

Общее свойство	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	пароль сервера	(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 имя пользователя не указано, оно будет запрошено у пользователя, при условии, что значение свойства user 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.

Свойство сервера	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 OS/400 V5R1 и более ранних версий.	нет	″″ (определять чувствительность курсора с помощью типа ResultSet) ″asensitive″ ″sensitive″ ″insensitive″	″″
″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 OS/400 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 пользовательского профайла. Если в описании задания не указано имя базы данных, применяется системная база данных по умолчанию.

Свойство сервера	Описание	Обязательное	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. В ряде случаев можно использовать значение *LIBL: • 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″
″maximum precision″	Задает максимальную точность операций с десятичными числами, которая может применяться базой данных.	нет	″31″ ″63″	″31″

Свойство сервера	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″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 uncommitted″ ″read committed″ ″repeatable read″ ″serializable″	″read uncommitted″
″translate hex″	Задает способ интерпретации шестнадцатеричных литералов.	нет	character (интерпретировать шестнадцатеричные литералы как символьные данные) binary (интерпретировать шестнадцатеричные литералы как двоичные данные)	″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″ (использовать истинную автоматическую фиксацию). ″false″ (не использовать истинную автоматическую фиксацию).	″false″

Свойства формата

Свойства формата задают формат и разделители даты и времени, а также соглашения о присвоении имен, применяемые с операторами SQL.

Свойство формата	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.

Свойство производительности	Описание	Обязательное	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 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 (записи не блокируются) 1 (блокируются, если указано значение FOR FETCH ONLY) 2 (блокируются, 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″
″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″

Свойство производительности	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″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	″″
″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″

Свойство производительности	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″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″
″qaqqinilib″	Задает имя библиотеки QAQQINI. С его помощью можно задать библиотеку, в которой расположен нужный файл qaqqini. Файл qaqqini содержит все атрибуты, от которых может зависеть производительность службы базы данных DB2 UDB for iSeries.	нет	″QAQQINI library name″	(значение сервера по умолчанию)

Свойства сортировки

Свойства сортировки управляют сохранением и сортировками.

Свойство сортировки	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″sort″	Указывает, каким образом сервер должен отсортировать записи перед их отправкой клиенту.	нет	″hex″ (сортировать с помощью шестн. значений) ″job″ (сортировать с помощью параметров задания сервера) ″language″ (сортировать с помощью значений языка в свойстве ″sort language″) ″table″ (сортировать на основе параметров таблицы сортировки свойства ″sort table″)	″job″
″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. Это свойство игнорируется, если свойству ″sort″ не присвоено значение ″language″.	нет	″shared″ (сортировка выполняется без учета регистра) ″unique″ (сортировка выполняется с учетом регистра)	″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	Обязательное	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″

Свойство	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″behavior override″	Указывает, какие алгоритмы работы драйвера JDBC IBM Toolbox for Java следует переопределить. Указав в этом свойстве сумму нескольких констант, вы можете изменить несколько алгоритмов. При этом необходимо обеспечить правильную обработку измененного алгоритма в вашем приложении.	нет	″″ (не переопределять алгоритмы) ″1″ (не создавать исключительных ситуаций и возвращать нулевое значение, если методы Statement.executeQuery() или PreparedStatement.executeQuery() не возвращают результатов)	″″
″bidi string type″	Задает тип строки вывода двунаправленных данных. Дополнительная информация приведена в разделе BidiStringType.	нет	″″ (определять тип двунаправленных данных с помощью CCSID) ″0″ (тип стандартных данных по умолчанию (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″
″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″
″driver″	Задает реализацию драйвера JDBC. Драйвер JDBC IBM Toolbox for Java в зависимости от среды может использовать различные реализации драйвера JDBC. В среде iSeries JVM, если база данных находится на локальном сервере, применяется драйвер JDBC IBM Developer Kit for Java. В других средах применяется драйвер JDBC IBM Toolbox for Java. Это свойство игнорируется, если не задано свойство ″secondary URL″.	нет	″toolbox″ (применять только драйвер JDBC IBM Toolbox for Java). ″native″ (применять на сервере драйвер JDBC IBM Developer Kit for Java, в других случаях - драйвер JDBC IBM Toolbox for Java).	″toolbox″

Свойство	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″errors″	Задает степень подробности сообщений об ошибках сервера.	нет	″basic″ ″full″	″basic″
″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 OS/400 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″

Свойство	Описание	Обязательное	Допустимые значения	Значение по умолчанию
″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″
″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″	″″
″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 системы OS/400)	″system″
″save password when serialized″	Указывает, нужно ли сохранять в локальной системе пароль вместе с остальными свойствами сериализованного исходного объекта. Сохранение пароля означает, что приложение должно обеспечить защиту сериализованного объекта, поскольку этот объект содержит всю информацию, необходимую для доступа к серверу. Перед присвоением значения true этому свойству оцените потенциальный риск, связанный с сохранением пароля вместе с остальными свойствами.	нет	″true″ ″false″	″false″

Свойство	Описание	Обяза-тельное	Допустимые значения	Значение по умолчанию
″secondary URL″	Задает URL для установления соединения с помощью драйвера DriverManager среднего уровня в многоуровневой среде. Это свойство позволяет подключиться с помощью этого драйвера к базам данных систем, отличных от iSeries. 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″
″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 выключена) ″2″ (запустить монитор базы данных для задания сервера JDBC) ″4″ (запустить отладку для задания сервера JDBC) ″8″ (сохранить протокол задания после завершения задания сервера JDBC) ″16″ (запустить трассировку задания для задания сервера JDBC) ″32″ (сохранить сведения SQL) 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″	Задает категорию трассировки IBM Toolbox for Java, которую необходимо заносить в протокол. 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 сведения о нитях)	″″

Свойство	Описание	Обязательное	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″

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.

Сценарий	Соглашение о присвоении имен SQL	Системное соглашение о присвоении имен
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
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

Неподдерживаемые типы SQL

Не все типы SQL, описанные в спецификации JDBC, поддерживаются DB2 для OS/400. 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
BIGINT (в OS/400 версии V4R4 и более ранних)	INTEGER
LONGVARCHAR	VARCHAR
LONGVARBINARY	VARBINARY

Примечание: BIGINT поддерживается OS/400 выпуска V4R5 и выше.

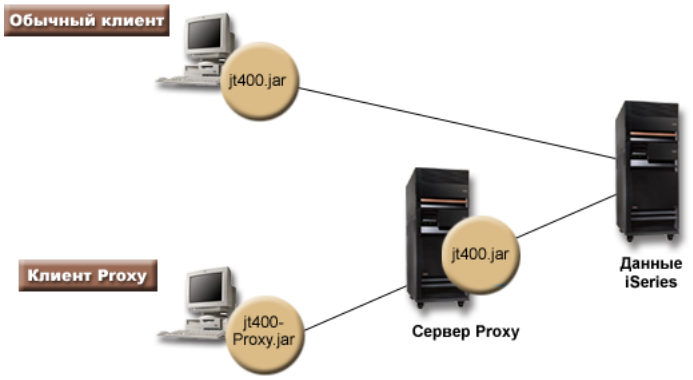
Поддержка Proxy

Некоторые классы IBM Toolbox for Java обеспечивают поддержку Proxy. Сервер Proxy позволяет выполнять задачи IBM Toolbox for Java на виртуальной машине JVM, отличной от той, где запускается приложение. В число средств для работы с сервером Proxy входит применение протокола SSL для шифрования данных.

Классы Proxy хранятся в файле jt400Proxy.jar и поставляются в составе IBM Toolbox for Java. Классы Proxy, как и другие классы IBM Toolbox for Java, представляют собой набор универсальных классов, работающих в любой системе, где установлена виртуальная машина Java. 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. В качестве сервера Proxy может выступать система iSeries, в которой хранятся данные.

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.

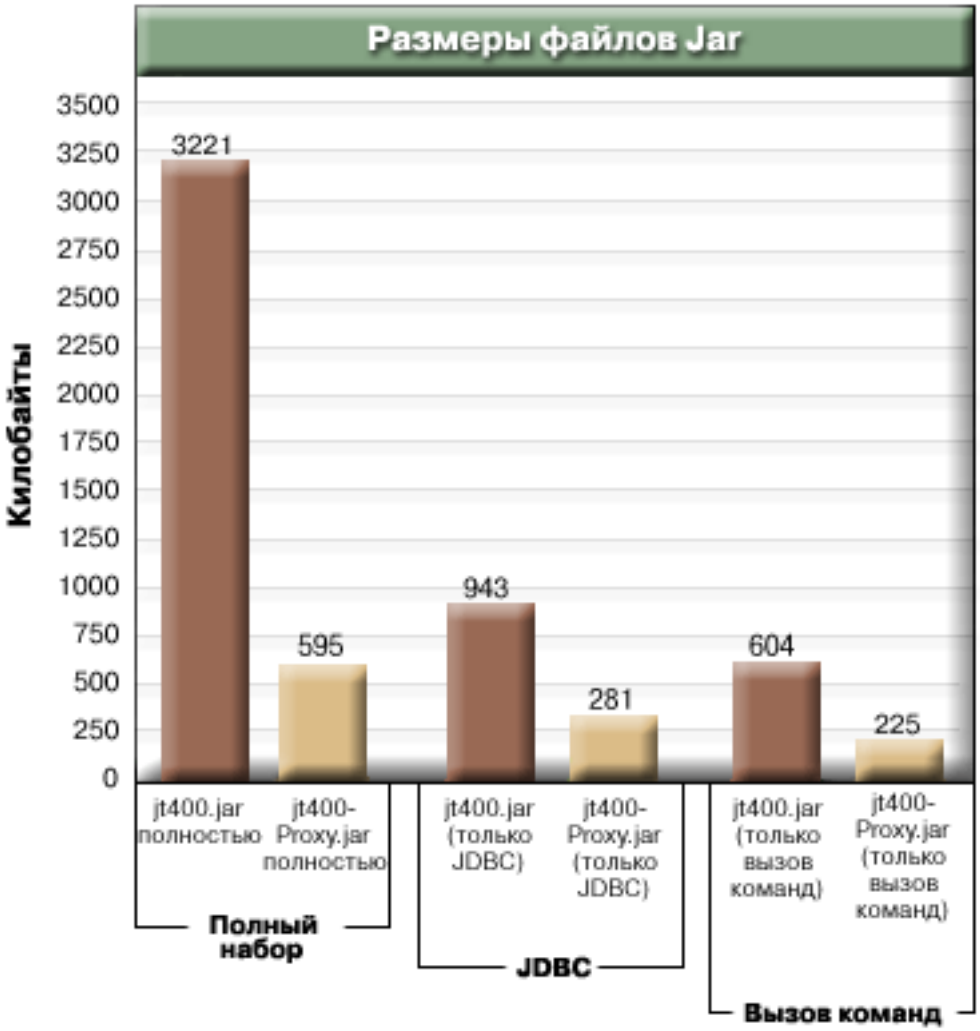
До появления поддержки Proxy классы, содержащие общий интерфейс, все классы обработки запросов и само приложение запускались на одной виртуальной машине Java (JVM). При использовании поддержки Proxy, классы общего интерфейса должны работать на одной виртуальной машине с приложением, а классы обработки 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. Определите, каким способом файл jt400Proxy.jar будет передаваться клиенту.
   - Если применяется программа на Java, то это можно сделать с помощью класса AS400ToolboxInstaller или другого метода передачи файла на клиент.
   - 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.

Применение SSL

При использовании Proxy существует три способа шифрования данных при их передаче с клиента на сервер iSeries. Шифрование данных выполняется с помощью алгоритмов SSL.

1. Возможно шифрование потоков данных, передаваемых между клиентом и сервером Proxy.
2. Возможно шифрование потоков данных, передаваемых между сервером Proxy и целевым сервером iSeries.
3. Оба вышеназванных варианта. Шифрование потоков данных, передаваемых между клиентом и сервером Proxy и между сервером Proxy и целевым сервером iSeries.

Дополнительная информация приведена в разделе Secure Sockets Layer.

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.

Подробное описание рисунка 1: Подключение к серверу обычного клиента и клиента Proxy (rzahh505.gif)

IBM Toolbox for Java: Поддержка серверов Proxy

Этот рисунок иллюстрирует:

- Подключение обычного клиента и клиента Proxy к серверу
- Необходимые файлы jar IBM Toolbox for Java

Описание

Рисунок состоит из следующих компонентов:

- Изображения вверху слева персонального компьютера, представляющего обычный клиент. Данное изображение включает круг с именем нужного файла Jar Toolbox for Java (jt400Proxy.jar).
- Изображения внизу слева персонального компьютера, представляющего клиент Proxy. Данное изображение включает круг с именем нужного файла Jar Toolbox for Java.
- Изображения посередине внизу сервера iSeries, представляющего сервер Proxy. Данное изображение включает круг с именем нужного файла Jar Toolbox for Java.
- Изображения справа сервера iSeries, представляющего сервер iSeries. Для работы сервера iSeries не требуется файл Jar, поэтому соответствующее изображение не включает круг.
- Линий, соединяющих изображения.

Обычный (стандартный) клиент назван на рисунке Сложным клиентом и использует файл jt400.jar. Обычный клиент подключается непосредственно к серверу iSeries.

Клиент Proxy использует файл jt400Proxy.jar. Клиент Proxy подключается к серверу Proxy, использующему файл jt400.jar. Сервер Proxy подключается к серверу iSeries.

Подробное описание рисунка 1: Сравнение размеров файлов jar proxy и стандартных файлов jar (rzahh502.gif)

IBM Toolbox for Java: Поддержка серверов Proxy

На этом рисунке изображена столбиковая диаграмма, позволяющая сравнить размеры стандартных файлов jar и файлов jar proxy, полученных после уменьшения размера файлов с помощью AS400JarMaker.

Описание

На диаграмме файл jt400.jar трижды сравнивается с файлом jt400Proxy.jar:

- Полные файлы jar
- Файлы jar, содержащие только компонент JDBC
- Файлы jar, содержащие только компонент Command Call

На столбиковой диаграмме изображены две оси: горизонтальная ось (ось x) и вертикальная ось (ось y).

- Горизонтальная ось диаграммы (или ось x) поделена на три группы столбцов, представляющих файлы jar.

- Вертикальная ось диаграммы (или ось y) отражает размер файла jar в килобайтах.

Группы носят имена Полный, JDBC и Command Call. Каждая группа содержит столбик для файла jt400.jar и столбик для файла jt400Proxy.jar. Ниже приведены описания групп:

- Полный: Размер полного файла jt400.jar составляет 3221 килобайт, размер полного файла jt400Proxy.jar составляет 595 килобайт.
- JDBC: Размер файла jt400.jar составляет 943 килобайта, а размер файла jt400Proxy.jar составляет 281 килобайт. Оба файла содержат только компонент JDBC.
- Command Call: Размер файла jt400.jar составляет 604 килобайта, а размер файла jt400Proxy.jar составляет 225 килобайт. Оба файла содержат только компонент Command Call.

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`. Необходимо, чтобы этот компьютер мог подключаться к системе iSeries.
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. Необходимо, чтобы этот компьютер мог подключаться к системе iSeries.
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 JVM : `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). **Примечание:** Убедитесь, что сервер HTTP подключен к системе iSeries, в которой расположены данные или ресурсы, используемые приложением, поскольку при выполнении запросов сервлет обращается к этой системе iSeries.
2. Выберите систему, которая будет выполнять роль клиента, и убедитесь, что переменная CLASSPATH в этой системе включает файл jt400Proxy.jar и файлы классов приложения. У клиента должна быть возможность подключения к серверу HTTP, но соединение с системой iSeries ему не требуется.
3. Задайте в качестве значения свойства com.ibm.as400.access.AS400.proxyServer имя сервера HTTP в формате URL.
4. Запустите приложение, задав в качестве значения свойства com.ibm.as400.access.AS400.proxyServer имя сервера HTTP в формате URL. 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 Java Secure Sockets Layer (SSL). JSSE поставляется в виде отдельного пакета продукта Java 2 Platform, Standard Edition (J2SE), версий 1.2 и 1.3. Продукт JSSE входит в состав продукта J2SE версии 1.4.

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).

После установки и настройки JSSE продукт IBM Toolbox for Java использует JSSE по умолчанию. Настоятельно рекомендуется перенести ваши существующие приложения на JSSE, поскольку дальнейшее обновление sslight не планируется. Инструкции по применению sslight по протоколу SSL приведены только для совместимости со старыми версиями продуктов.

Перед началом применения SSL с IBM Toolbox for Java ознакомьтесь со своими правовыми ограничениями.

IBM Toolbox for Java 2 Micro Edition

Пакет IBM Toolbox for Java 2 Micro Edition (com.ibm.as400.micro) позволяет создавать программы на Java, с помощью которых различные беспроводные устройства Tier0, например электронные записные книжки (PDA) и сотовые телефоны, могут напрямую обращаться к данным и ресурсам iSeries.

Дополнительная информация о ToolboxME for iSeries приведена в следующих разделах:

Требования
Информация о необходимых условиях для разработки приложений с помощью ToolboxME for iSeries и запуска этих приложений на устройствах Tier0.

Загрузка и настройка ToolboxME for iSeries
Инструкции по загрузке и настройке ToolboxME for iSeries на сервере, рабочей станции и устройстве Tier0.

Общие сведения
Краткое описание основных принципов разработки приложений, предназначенных для устройств Tier0.

Классы ToolboxME for iSeries
Перечень классов, входящих в компонент ToolboxME for iSeries (пакет com.ibm.as400.micro). Эти классы содержат сокращенный набор функций классов доступа Toolbox for Java, поддержки JDBC и пр.

Создание программы ToolboxME for iSeries
Пошаговые инструкции по созданию программ ToolboxME for iSeries для устройств Tier0. Выполнив инструкции, вы создадите свою первую программу ToolboxME for iSeries.

Примеры
Просмотрите, загрузите и запустите примеры программ ToolboxMe for iSeries, которые помогут вам создавать и применять приложения для беспроводных устройств.

Загрузка и настройка ToolboxME for iSeries

Вы должны отдельно загрузить компонент ToolboxME for iSeries (jt400Micro.jar), содержащийся в JTOpen.

ToolboxME for iSeries можно загрузить с Web-сайта IBM Toolbox for Java/JTOpen  , который также содержит дополнительную информацию о настройке ToolboxME for iSeries.

Настройка ToolboxME for iSeries на устройстве Tier0, на рабочей станции и на сервере выполняется по-разному:

- Создайте приложение для беспроводного устройства (с помощью файла jt400Micro.jar), затем установите приложение согласно инструкциям производителя устройства.
- Убедитесь, что на сервере, содержащем целевые данные, запущены серверы хоста iSeries.
- 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. 13

“Установка IBM Toolbox for Java на сервере iSeries” на стр. 12

Важная информация о работе с ToolboxME for iSeries

Перед тем, как вы приступите к разработке приложений на Java с помощью ToolboxME for iSeries, ознакомьтесь с приведенными ниже принципами и стандартами, которыми следует руководствоваться при разработке.

Java 2 Platform, Micro Edition (J2ME)

J2ME(TM) - это реализация стандарта Java 2, предоставляющего среды выполнения Java для беспроводных устройств Tier0, таких как электронные записные книжки (PDAs) 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.

Дополнительная информация приведена в разделе 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:

- Виртуальная машина IBM J9, входящая в состав 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. Краткий перечень таких средств приведен в разделе IBM Toolbox for Java - Связанная информация.

Дополнительную информацию и загружаемые симуляторы и эмуляторы беспроводных устройств вы найдете на Web-сайте, посвященном устройству или операционной системе, для которой предназначено ваше приложение.

Классы ToolboxME for iSeries

Пакет com.ibm.as400.micro содержит классы, необходимые для создания приложений, позволяющих подключаться к данным и ресурсам сервера iSeries с помощью “Устройства Tier0” на стр. 351.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries.

В ToolboxME for iSeries входят следующие классы:

- “Класс MEServer” передает запросы от устройства Tier0 на сервер хоста
- Несколько классов предоставляют набор функций из пакета доступа Toolbox for Java:
  - “Класс AS400” на стр. 353 позволяет входить на сервер iSeries
  - “Класс CommandCall” на стр. 353 позволяет вызывать команды iSeries
  - “Класс DataQueue” на стр. 354 позволяет читать и записывать данные в очередь данных сервера iSeries
  - “Класс ProgramCall” на стр. 355 позволяет вызывать программы iSeries и получать выходные данные этих программ
- Классы “Классы JdbcMe” на стр. 356 обеспечивают поддержку JDBC, предоставляя минимальный набор методов и данных пакета java.sql

Класс MEServer

Класс MEServer позволяет выполнять запросы клиентского приложения Tier0, применяющего jar-файл ToolboxME for iSeries. От имени клиентского приложения класс MEServer создает объекты IBM Toolbox for Java и применяет к ним методы.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе Загрузка и настройка ToolboxME for iSeries.

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`.

Важная информация о применении этой опции приведена в разделе 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.

Класс AS400

Класс AS400 в пакете micro (com.ibm.as400.micro.AS400) предоставляет модифицированный набор функций класса AS400 в пакете доступа (com.ibm.as400.access.AS400). С помощью класса ToolboxMe for iSeries AS400 class войдите на сервер iSeries с помощью устройства Tier0.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе Загрузка и настройка ToolboxME for iSeries.

Класс AS400 позволяет:

- Подключиться к серверу MEServer
- Отключиться от сервера MEServer

Соединение с сервером MEServer устанавливается неявно. Например, если после создания объекта AS400 вы запустите метод run() в CommandCall, то метод connect() будет выполнен автоматически. Иными словами, вам не требуется явно вызывать метод connect(), если только вы не хотите проконтролировать установление соединения.

Пример: Применение класса AS400

Ниже приведен пример входа на сервер iSeries с помощью класса AS400:

```
AS400 system = new AS400("mySystem", "myUserid", "myPwd", "myMEServer");
try
{
    system.connect();
}
catch (Exception e)
{
    // Обработка исключительной ситуации
}
// Обработка системного объекта завершена.
system.disconnect();
```

Класс CommandCall

Класс CommandCall в пакете micro (com.ibm.as400.micro.CommandCall) предоставляет модифицированный набор функций класса CommandCall в пакете доступа (com.ibm.as400.access.CommandCall). Класс CommandCall позволяет вызывать команду iSeries из устройства Tier0.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries.

Аргументом метода run() класса CommandCall служит Строка (запускаемая команда); метод возвращает все сообщения, выдаваемые в результате выполнения этой команды. 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();
```

Класс DataQueue

Класс DataQueue в пакете micro (com.ibm.as400.micro.DataQueue) предоставляет модифицированный набор функций класса DataQueue в пакете доступа (com.ibm.as400.access.DataQueue). С помощью класса DataQueue вы можете организовать обмен информацией между устройством Tier0 и очередью данных на сервере iSeries.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries.

В класс DataQueue входят следующие методы:

- Чтение и запись отдельной записи как строки
- Чтение и запись отдельной записи как массива байтов

Для чтения или записи информации вы должны указать имя системы iSeries, в которой находится очередь данных, и полное имя очереди данных в интегрированной файловой системе. Если информации нет, то результатом чтения будет пустое значение.

Пример: Чтение и запись в очередь данных с помощью класса DataQueue

Ниже приведен пример применения класса DataQueue для чтения и записи в очередь данных в системе iSeries:

```
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();
```

Класс ProgramCall

Класс ProgramCall в пакете micro (com.ibm.as400.micro.ProgramCall) предоставляет модифицированный набор функций класса ProgramCall в пакете доступа (com.ibm.as400.access.ProgramCall). Класс ProgramCall позволяет устройству Tier0 вызвать программу iSeries и получить доступ к результатам ее выполнения.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе “Требования ToolboxME for iSeries” на стр. 10.

Для применения метода ProgramCall.run() необходимо указать следующие параметры:

- Система, в которой следует запустить программу
- Имя документа “Язык описания вызовов программ” на стр. 378
- 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.

Дополнительная информация о хэш-таблице, содержащей параметры программы, и о регистрации документа PCML приведены в разделе ToolboxME for iSeries ProgramCall javadoc. Дополнительная информация о PCML приведена в разделе “Язык описания вызовов программ” на стр. 378.

Пример: Применение объекта ProgramCall

Ниже приведен пример использования класса ProgramCall для запуска программы на сервере с помощью устройства Tier 0:

```
// 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();
```

Классы JdbcMe

Классы ToolboxME for iSeries предоставляют поддержку JDBC, включая поддержку пакета java.sql. Эти классы предназначены для использования в программе, выполняемой на устройстве Tier 0.

В следующих разделах рассмотрен доступ к данным и их использование и описано содержимое JdbcMe, включая ссылки на информацию об отдельных классах 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.

Для того чтобы упростить работу с моделью синхронизации данных, классы JdbcMe в ToolboxME for iSeries позволяют выполнять обновление данных и обращаться к главной базе данных в режиме прямого доступа, сохраняя вместе с тем возможность автономного хранения данных. 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

В JdbcMe входят следующие классы:

- JdbcMeConnection
- JdbcMeDriver
- JdbcMeException
- JdbcMeLiveResultSet
- JdbcMeOfflineData
- JdbcMeOfflineResultSet
- JdbcMeResultSetMetaData
- JdbcMeStatement

В ToolboxME for iSeries предусмотрен пакет java.sql, соответствующий спецификации JDBC, но содержащий минимальный набор полезных классов и методов. Благодаря этому классы JdbcMe имеют малый размер, при этом позволяя выполнять обычные задачи JDBC.

Подключение к базе данных сервера хоста с помощью драйвера Toolbox ME for iSeries:

Класс JdbcMeConnection содержит часть функций класса AS400JDBCConnection IBM Toolbox for Java. С помощью JdbcMeConnection вы можете предоставить устройству Tier0 доступ к базам данных DB2 Universal Database (UDB) на сервере хоста.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе Требования и установка ToolboxME for iSeries.

Для подключения к базе данных сервера воспользуйтесь методом JdbcMeDriver.getConnection(). 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://имя-сервера/схема-по-умолчанию;meserver=<сервер>[:порт];[другие свойства];
```

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();
```

Класс JdbcMeDriver:

Класс JdbcMeDriver содержит часть функций класса AS400JDBCDriver IBM Toolbox for Java. Класс JdbcMeDriver в клиентском приложении Tier0 позволяет выполнять простые операторы SQL без параметров и получать выдаваемые ими наборы результатов.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе “Загрузка и настройка ToolboxME for iSeries” на стр. 350.

Явная регистрация JdbcMeDriver не выполняется; драйвер определяется с помощью свойства **driver**, указываемого в URL в методе JdbcMeConnection.getConnection(). Например, для загрузки драйвера IBM Developer Kit for Java JDBC (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 Toolbox for Java, получающих информацию с сервера, драйвер JDBC 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.

Наборы результатов:

Ниже перечислены классы наборов результатов ToolboxME for iSeries:

- 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: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе Загрузка и настройка ToolboxME for iSeries.

JdbcMeLiveResultSet

Класс JdbcMeLiveResultSet содержит часть функций класса AS400JDBCResultSet IBM Toolbox for Java. 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:

- Получить данные различных типов из набора результатов
- Переместить курсор в указанную строку (предыдущую, текущую, следующую и т.п.)
- Вставить, обновить или удалить строки
- Обновить столбцы (с помощью строковых и целочисленных значений)
- Получить объект ResultSetMetaData, содержащий описание столбцов набора результатов

Курсор - это внутренний указатель на строку таблицы результатов, к которой в данный момент обратилась программа на Java. JDBC 2.0 предоставляет дополнительные методы для перемещения по базе данных:

Перемещение курсора путем прокрутки
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 JDBC 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.

Полный список методов обновления класса JdbcMeResultSet приведен в разделе Обзор методов.

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

Класс JdbcMeOfflineResultSet содержит часть функций класса AS400JDBCResultSet IBM Toolbox for Java. Класс JdbcMeOfflineResultSet в клиентском приложении Tier0 позволяет получить доступ к таблице данных, созданной в результате выполнения запроса.

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:

- Получить данные различных типов из набора результатов
- Переместить курсор в указанную строку (предыдущую, текущую, следующую и т.п.)
- Вставить, обновить или удалить строки
- Обновить столбцы (с помощью строковых и целочисленных значений)
- Получить объект ResultSetMetaData, содержащий описание столбцов набора результатов

С помощью функций классов JdbcMe вы можете обеспечить синхронизацию базы данных локального устройства с базой данных сервера iSeries.

Класс JdbcMeResultSetMetaData

Класс JdbcMeResultSetMetaData содержит часть функций класса AS400JDBCResultSetMetaData IBM Toolbox for Java. 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();
```

Класс JdbcMeOfflineData:

Класс JdbcMeOfflineData - это автономное хранилище данных на устройстве Tier0. Хранилище данных не зависит от применяемого профайла и виртуальной машины Java. Дополнительная информация приведена в разделе ToolboxME for iSeries - Принципы.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе Загрузка и настройка ToolboxME for iSeries.

Методы класса JdbcMeOfflineData позволяют выполнить следующие действия:

- Создать автономное хранилище данных
- Открыть существующее хранилище
- Получить число записей в хранилище
- Получить или удалить отдельные записи
- Обновить записи
- Добавить запись в конец хранилища
- Закрыть хранилище

Пример применения класса JdbcMeOfflineData приведен в следующем разделе:

“Пример: Применение ToolboxME for iSeries, MIDP и IBM Toolbox for Java” на стр. 694

Класс JdbcMeStatement:

Класс JdbcMeStatement содержит часть функций класса AS400JDBCStatement IBM Toolbox for Java. Класс JdbcMeStatement t в клиентском приложении Tier0 позволяет выполнять простые операторы SQL без параметров и получать выдаваемые ими наборы результатов.

Примечание: Для применения классов ToolboxMe for iSeries необходимо специально загрузить и настроить компонент ToolboxME for iSeries. Дополнительная информация приведена в разделе Загрузка и настройка ToolboxME for iSeries.

Для создания объектов Statement предназначен метод JdbcMeConnection.createStatement().

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();
```

Создание и запуск программ ToolboxME for iSeries

Данная информация содержит инструкции по редактированию, компиляции и запуску примера программы ToolboxME for iSeries. Кроме того, эту информацию можно рассматривать в качестве общего руководства по созданию, тестированию и запуску рабочих примеров ToolboxME for iSeries и пользовательских приложений ToolboxME for iSeries.

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).

Перед тем, как вы приступите к созданию примеров программ ToolboxME for iSeries, убедитесь, что среда удовлетворяет требованиям ToolboxME for iSeries.

Создание примера программы ToolboxME for iSeries

Для создания примера программы ToolboxME for iSeries для устройства Tier0 выполните следующие действия:

1. Скопируйте код Java примера программы ToolboxME for iSeries. Это файл 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.

Информация о принципах работы с продуктом, средствах разработки приложений поддержки беспроводных устройств и эмуляторах приведена в разделе ToolboxME for iSeries - Принципы.

Запуск примера программы ToolboxME for iSeries

Для запуска примера программы ToolboxME for iSeries на устройстве Tier0 выполните следующие действия:

- Загрузите исполняемый файл на устройство Tier0 согласно инструкциям производителя устройства.
- Запустите сервер MEServer
- Запустите программу JdbcDemo на устройстве Tier0, щелкнув на значке JdbcDemo.

Пример программы ToolboxME for iSeries: JdbcDemo.java

Для преобразования данного примера в рабочую программу ToolboxME for iSeries вы должны скопировать следующий файл .java в окно текстового редактора или редактора Java, внести некоторые изменения, а затем откомпилировать файл.

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.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример программы ToolboxME for iSeries. Программа демонстрирует, каким образом
// беспроводное устройство может подключиться к серверу iSeries и с помощью JDBC
// 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
        // MEServer и сервер iSeries, к которому вы подключаетесь.
        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);
        }
    }
}
```

Примеры программ ToolboxME for iSeries

Приведенные ниже примеры для ToolboxMe for iSeries демонстрируют возможности применения ToolboxME for iSeries с “Профайл мобильных устройств (MIDP)” на стр. 351. Для просмотра выбранных исходных файлов или загрузки всех исходных файлов, необходимых для создания приложений поддержки беспроводных устройств, выберите одну из следующих ссылок:

“Пример: Применение ToolboxME for iSeries, MIDP и JDBC” на стр. 685

“Пример: Применение ToolboxME for iSeries, MIDP и IBM Toolbox for Java” на стр. 694

“Загрузка примеров для ToolboxME for iSeries”

Дополнительная информация о создании приложений ToolboxME for iSeries приведена в разделе “Создание и запуск программ ToolboxME for iSeries” на стр. 363.

Загрузка примеров для ToolboxME for iSeries

Для создания приложений, предназначенных для работы с беспроводными устройствами, на основе примеров применения ToolboxME for iSeries вам необходимы все исходные файлы, а также дополнительные инструкции. 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 “Создание и запуск программ ToolboxME for iSeries” на стр. 363 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:

- “Требования ToolboxME for iSeries” на стр. 10
- “Загрузка и настройка ToolboxME for iSeries” на стр. 350

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
- Вызов программ в системе iSeries и получение результатов программ
- Указание форматов данных в системе iSeries

Дополнительная информация приведена на следующих страницах:

PCML
Информация о работе с Языком описаний вызовов программ (PCML). Применение PCML для вызова программ iSeries позволяет сократить объем кода Java. PCML - это язык тегов, полностью описывающий входные и выходные параметры программ iSeries, вызываемых приложением на Java.

Язык PDML
Язык PDML, который входит в состав Graphical Toolbox, может применяться на любых платформах и служит для описания расположения элементов пользовательского интерфейса. После создания

определений панелей в PDML их можно просмотреть с помощью API выполнения Graphical Toolbox. При выводе панелей этот API интерпретирует описание PDML и отображает элементы пользовательского интерфейса с помощью классов Java Foundation.

RFML
Применение RFML позволяет отделить спецификации форматов данных от алгоритмов в программах на Java. RFML - это язык тегов, основанный на XML и тесно связанный с PCML. С помощью PCML вы можете определять поля и управлять ими в записях определенного типа в приложениях на Java.

Анализатор XML и обработчик XSLT
В этом разделе приведена информация об анализаторе XML и обработчике XSLT, которые необходимы для применения компонентов XML IBM Toolbox for Java.

Язык XPCML
Язык XPCML расширяет функциональные возможности и сферу применения PCML благодаря поддержке схем XML. XPCML обладает несколькими преимуществами, включая возможность задавать и передавать значения параметров программ и получать результаты вызова программ в системе iSeries в формате XPCML.

Язык описания вызовов программ

Язык описания вызовов программ (PCML) - это язык тегов, позволяющий создавать вызовы программ сервера, что сокращает объем кода Java. 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.

Дополнительная информация о применении PCML приведена на следующих страницах:

- “Создание вызовов программ iSeries с помощью PCML”
- “Синтаксис PCML” на стр. 383
- “Примеры кода на Языке описаний вызовов программ (PCML)” на стр. 607

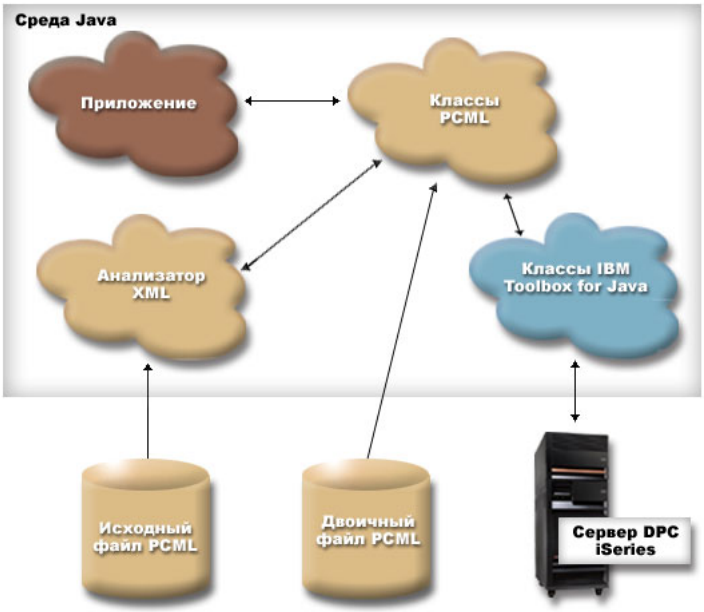
Создание вызовов программ iSeries с помощью PCML

Перед разработкой вызовов программ iSeries на языке PCML нужно создать приложение на Java и исходный файл PCML.

В зависимости от структуры программы, создайте один или несколько исходных файлов PCML с описанием интерфейсов для работы с программами iSeries. Эти интерфейсы будут вызываться приложением на Java. Подробное описание языка приведено в разделе Синтаксис PCML.

Приложение на Java взаимодействует с классами PCML (в данном случае - с классом ProgramCallDocument). Класс ProgramCallDocument применяет исходный файл PCML для передачи информации из приложения на Java в систему iSeries и обратно. 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.

Приложение применяет методы созданного класса ProgramCallDocument для получения необходимой информации из системы iSeries с помощью сервера вызовов распределенных программ (DPC) AS/400.

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 Java или метода setPath() класса ProgramCallDocument. В случае, когда в приложении на Java необходимо задавать путь к файлу PCML во время выполнения, следует воспользоваться методом setPath().

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, 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

Для задания входных параметров вызываемой программы iSeries в приложениях на Java применяется метод ProgramCallDocument.setValue(). Для получения выходных значений программы iSeries в приложениях применяется метод ProgramCallDocument.getValue().

При обращении к значениям класса ProgramCallDocument необходимо указывать полное имя элемента документа или тега **<data>**. 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">
    <!-- Параметр 1 содержит массив и число многоугольников -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Каждый многоугольник содержит число вершин и массив с их координатами -->
      <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

Все элементы **<data>** и **<struct>** можно определить в виде массивов с помощью атрибута **count**. Кроме того, элементы **<data** и **<struct** могут содержаться внутри другого элемента **<struct**, также являющегося массивом.

Более того, элементы **<data** и **<struct** могут содержаться в многомерном массиве, если сразу для нескольких внутренних элементов задан атрибут **count**.

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:

- Дамп шестнадцатеричных данных, которыми обмениваются приложение на Java и программа iSeries. 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 шестнадцатеричного 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
```

Такие сообщения чрезвычайно полезны для выявления случаев, когда результат работы программы iSeries не соответствует исходному коду PCML. Это часто происходит при работе с динамическими массивами данных и смещениями.

Подробное описание рисунка 1: вызов программ сервера с помощью PCML (rzahh503.gif):

IBM Toolbox for Java: Обработка PCML

Этот рисунок демонстрирует применение программы на Java совместно с классами PCML.

Описание

Рисунок поделен на две области: верхняя представляет среду Java, а нижняя - часть процесса PCML, не имеющую отношения к Java.

- В верхней области рисунка (среде Java) схематично изображены ″Приложение,″ ″Классы PCML,″ ″Классы доступа IBM Toolbox for Java″ и ″Анализатор XML.″
- В нижней области рисунка (другой части процесса) схематично изображены ″Источник PCML,″ ″Двоичный формат PCML″ и ″Сервер вызовов распределенных программ (DPC) iSeries.″
- Изображения соединяют однонаправленные или двунаправленные стрелки. Двунаправленная стрелка говорит о том, что оба объекта взаимодействуют друг с другом. Однонаправленная стрелка указывает на объект, который использует второй объект.

Приложение на Java взаимодействует с классами PCML. В этом примере приложение создает объект ProgramCallDocument.

После создания объекта ProgramCallDocument выполняется одно из следующих действий:

- Анализатор XML обрабатывает исходные файлы PCML и передает информацию классам PCML. В исходных файлах PCML описаны интерфейсы для доступа к программам iSeries, вызываемым из приложения на Java.
- Информация PCML передается классам PCML в двоичной форме. Применение информации PCML, предварительно преобразованной в двоичный формат, позволяет повысить производительность, так как во время выполнения программы не требуется анализировать данные PCML. Исходный код PCML можно преобразовать в двоичный формат во время создания приложения.

Классы PCML также взаимодействуют с классами Toolbox for Java, которые в данном примере получают информацию с сервера с помощью сервера вызова распределенных программ iSeries.

Указанные взаимосвязи определяют способ передачи информации между приложением на Java и программами iSeries.

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.
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().

Атрибут	Значение	Описание
parseorder=	*список-имен*	Задает порядок обработки выходных параметров. Значением должен быть список имен параметров, разделенных пробелами. Порядок имен определяет последовательность их обработки. В списке должны быть заданы те же имена, что и в атрибуте **name** тегов объекта **<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.
threadsafe=	*true* Программа поддерживает нити. *false* Программа не поддерживает нити.	При вызове программы на Java и программы iSeries в одной системе это свойство определяет, будет ли программа iSeries вызвана в том же задании и в той же нити, что и программа на Java. 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={ наследование | ввод | вывод | ввод-вывод } ]>
</struct>
```

В следующей таблице перечислены атрибуты тега структуры. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
name=	*имя*	Задает имя элемента **<struct>**
count=	*Число*, где *число* задает фиксированный неизменяемый массив с конечным числом элементов. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента а **<data>** документа PCML, который при выполнении программы будет содержать число элементов массива. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Указывает, что элемент является массивом указанного размера. Если атрибут count не задан, то элемент не является массивом, хотя он может быть элементом массива.
maxvrm=	*версия*	Задает максимальную версию OS/400, в которой поддерживается этот элемент. Если версия OS/400 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 OS/400. 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*	Задает минимальную версию OS/400, в которой поддерживается этот элемент. Если версия OS/400 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 OS/400. 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.
offset=	*Число*, где *число* определяет фиксированное неизменяемое смещение. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа PCML, который при выполнении программы будет содержать смещение элемента. **Имя-элемента-данных** может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает смещение элемента **<struct** в выходном параметре. 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. Атрибут **offset** служит для указания смещения элемента **<struct>**. 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** служат только для обработки выходных данных 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. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа PCML, содержащего основное расположение, от которого определяется смещение элемента. 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. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен. *Имя-структуры*, где *имя-структуры* определяет имя элемента **<struct>**, задающего основное расположение данных, относительно которого определяется смещение. 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. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает точку отсчета смещения, заданного в атрибуте **offset**. 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** служат только для обработки выходных данных программы. Эти атрибуты не задают смещение ввода.

Атрибут	Значение	Описание
outputsize=	*Число*, где *число* задает фиксированное неизменяемое число резервных байтов. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа PCML, который при выполнении программы будет содержать число резервных байтов для выходных данных. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	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. Если этот атрибут не задан, объем памяти, резервируемый для вывода, определяется во время выполнения путем сложения значений этого атрибута, заданных для всех потомков элемента **<struct**.
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">
    <!-- переменная receiver содержит полный путь -->
    <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">
    <!-- переменная receiver содержит объект -->
    <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={ наследование | ввод | вывод | ввод-вывод } ]>
</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[]*). Дополнительная информация приведена в разделе Указание длины данных с помощью значений *byte* . *struct*, где *struct* задает имя элемента **<struct>**. Объект *struct* позволяет определить структуру и затем несколько раз 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.	Задает тип двунаправленных строк для элементов **<данных>**, для которых задан атрибут **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. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает CCSID символьных данных для элемента **<data>**. Атрибут **ccsid** можно задать только для элементов **<data>** с атрибутом **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.

Атрибут	Значение	Описание
count=	*number*, где *number* определяет фиксированное неизменяемое число элементов конечного массива. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа PCML, который при выполнении программы будет содержать число элементов массива. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае, имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Указывает, что элемент является массивом указанного размера. Если атрибут *count* не задан, то элемент не является массивом, хотя он может быть элементом массива.
init=	*строка*	Задает начальное значение элемента **<data>**. Значение *init* применяется, если в программе начальное значение не задано явно и в ней используются элементы **<data>** с атрибутом **usage=***"input"* или **usage=***"inputoutput"*. 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. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа PCML, который при выполнении программы будет содержать значение длины данных. *Имя-элемента-данных* можно задать только для элементов **<data>** с атрибутом **type=***"char"* или **type=***"byte"*. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает длину элемента данных. Назначение этого атрибута зависит от типа данных. Дополнительная информация приведена в разделе Значения длины и точности.

Атрибут	Значение	Описание
maxvrm=	*версия*	Задает максимальную версию системы iSeries, в которой поддерживается этот элемент. Если версия iSeries 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 iSeries. 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 iSeries, в которой поддерживается этот элемент. Если версия iSeries меньше 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 iSeries. 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*	Задает имя элемента **<data>**.

Атрибут	Значение	Описание
offset=	*Число*, где *число* определяет фиксированное неизменяемое смещение. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа PCML, который при выполнении программы будет содержать смещение данного элемента. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает смещение элемента **<data** в выходном параметре. 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. 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. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>**, соответствующего основному расположению данных, относительно которого определяется смещение. 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. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен. *Имя-структуры*, где *имя-структуры* определяет имя элемента **<struct>**, задающего основное расположение данных, относительно которого определяется смещение. 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. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает точку отсчета смещения, заданного в атрибуте **offset**. 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** служат только для обработки выходных данных программы. Эти атрибуты не задают смещение ввода.

Атрибут	Значение	Описание
outputsize=	*Число*, где *число* задает фиксированное неизменяемое число резервных байтов. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа PCML, который при выполнении программы будет содержать число резервных байтов для выходных данных. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=***int*. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	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 iSeries. 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. Если атрибут **outputsize** не указан, число байтов, зарезервированных для выходных данных, определяется при выполнении программы. Окончательным значением будет сумма всех соответствующих значений дочерних объектов элемента **<struct>**.
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. Данный атрибут является допустимым, только если этот элемент является дочерним объектом элемента **<program>**, определяющего вызов служебной программы.
precision=	*число*	Задает число значимых разрядов для некоторых числовых типов данных. Дополнительная информация приведена в разделе Значения длины и точности.
struct=	*имя*	Задает имя элемента **<struct** в элементе **<data**. Атрибут **struct** можно задать только для элементов **<data>** с атрибутом **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">
    <!-- переменная receiver содержит полный путь -->
    <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">
    <!-- переменная receiver содержит объект -->
    <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. При этом в итоге должен быть найден тег, который содержится в теге **<pcml>** или теге **<rfml>**; в этом случае имя является абсолютным, а не относительным.

Ниже приведен пример применения PCML:

```
<pcml version="1.0">
  <program name="polytest" path="/QSYS.lib/MYLIB.lib/POLYTEST.pgm">
    <!-- Параметр 1 содержит массив и число многоугольников -->
    <struct name="parm1" usage="inputoutput">
      <data name="nbrPolygons" type="int" length="4" init="5" />
      <!-- Каждый многоугольник содержит число вершин и массив их координат -->
      <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">
    <!-- Этот параметр позволяет размещать число и массив многоугольников -->
    <data name="nbrPolygons" type="int" length="4" init="5" />
    <data name="polygon" type="struct" struct="polygon" count="nbrPolygons" />
  </recordformat>
</rfml>
```

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.

Документы RFML, которые называются исходными файлами RFML, представляют функциональный набор типов спецификаций описания данных (DDS) для физических и логических файлов в системах iSeries. Документы RFML применяются для управления 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 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.

Дополнительная информация о RFML приведена в следующих разделах:

Требования
Требования, которые должны быть выполнены для применения RFML.

Пример RFML
Этот пример демонстрирует, как применение RFML при создании приложения позволяет снизить объем, а в некоторых случаях - и сложность создаваемого исходного кода. Пример содержит исходный файл RFML.

Класс RecordFormatDocument
Применение класса RecordFormatDocument совместно с другими классами продукта Toolbox for Java для чтения и записи данных.

Документы RFML и синтаксис RFML
Информация о документах RFML, или исходных файлах RFML, и о синтаксисе RFML, описанном в определении типов данных RFML.

RFML - это единственный способ применения XML на сервере. Дополнительная информация о применении XML в системе iSeries приведена в документации к расширениям XML IBM Toolbox for Java и разделе Язык XML.

Требования для применения RFML

Компонент RFML предъявляет те же требования к виртуальной машине Java рабочей станции, что и прочие компоненты IBM Toolbox for Java.

Помимо этого, для динамического анализа RFML необходимо, чтобы в переменной CLASSPATH приложения был указан анализатор XML. Анализатор XML должен расширять класс org.apache.xerces.parsers.SAXParser. Дополнительная информация приведена на следующей странице:

“Анализатор XML и обработчик XSLT” на стр. 412

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.

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 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

Дополнительная информация о методах данного класса находится в документе Обзор методов класса RecordFormatDocument.

Применение класса RecordFormatDocument с другими классами IBM Toolbox for Java

Класс RecordFormatDocument можно применять вместе со следующими классами 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 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.
- Классы JDBC, поскольку RFML работает только с данными, описанными в спецификациях описания данных (DDS) iSeries.

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.

Пример Сравнение RFML с классами Record продукта Toolbox for Java содержит пример исходного файла RFML.

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. Теги RFML предназначены для объявления и определения следующих элементов исходных файлов RFML:

- Тег rfml обозначает начало и конец исходного файла RFML, описывающего формат данных.
- Тег struct определяет именованную структуру, которую можно повторно использовать в исходном файле RFML. Для каждого поля структуры задается тег data.
- Тег recordformat определяет формат записи, содержащей элементы данных или ссылки на элементы структуры.
- Тег data определяет поле в формате записи или структуре.

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 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.
-->

<!-- Стандартные объекты -->
<!ENTITY % string             "CDATA">    <!-- строка нулевой или ненулевой длины -->
<!ENTITY % nonNegativeInteger "CDATA">    <!-- неотрицательное целое число -->
<!ENTITY % binary2            "CDATA">    <!-- целое число от 0 до 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)">


<!-- Корневой элемент документа -->
<!ELEMENT rfml (struct | recordformat)+>
<!ATTLIST rfml
          version       %string;    #FIXED "4.0"
          ccsid         %binary2;             #IMPLIED
>
<!-- Примечание: Значение ccsid будет по умолчанию применяться для интерпретации -->
     <!-- любых <data type="char"> элементов, для которых не задан 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. -->


<!-- Стандартные предопределенные объекты символов -->
<!ENTITY quot  "&#34;">     <!-- кавычки -->
<!ENTITY amp   "&#38;#38;"> <!-- кавычки -->
<!ENTITY apos  "&#39;">     <!-- апостроф -->
<!ENTITY lt    "&#38;#60;"> <!-- знак меньше -->
<!ENTITY gt    "&#62;">     <!-- знак больше -->
<!ENTITY nbsp  " ">    <!-- неразрывный пробел -->
<!ENTITY shy   "&#173;">    <!-- мягкий дефис (раздельный дефис) -->
<!ENTITY mdash "&#38;#x2014;">
<!ENTITY ldquo "&#38;#x201C;">
<!ENTITY rdquo "&#38;#x201D;">
```

Тег данных RFML:

В теге данных предусмотрены 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 }" ]
    [ count="{ число | имя-элемента-данных }" ]
    [ init="string" ]
    [ length="{ число | имя-элемента-данных }" ]
    [ name="имя" ]
    [ precision="число" ]
    [ struct="имя-структуры" ]>
</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* Имя элемента **<struct>**. 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*, 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.	Задает тип двунаправленных строк для элементов **<data>**, для которых задан атрибут **type=**`char`. Если этот атрибут не задан, то тип строки определяется с помощью явно заданного CCSID или CCSID хоста по умолчанию. Строчные типы описаны 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. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=**`int`. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает CCSID символьных данных для элемента **<data>**. Атрибут **ccsid** можно задать только для элементов **<data>** с атрибутом **type=**`char`. Если этот атрибут не задан, то считается, что CCSID символьных данных совпадает с CCSID хоста по умолчанию.

Атрибут	Значение	Описание
count=	*Число*, где *число* определяет фиксированное неизменяемое число элементов конечного массива. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента а **<data>** документа RFML, который при выполнении программы будет содержать количество элементов массива. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=**`int`. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Указывает, что элемент является массивом указанного размера. Если атрибут `count` не задан, то элемент не является массивом, хотя он может быть элементом массива.
init=	*строка*	Задает начальное значение элемента **<data>**. 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. *Имя-элемента-данных*, где *имя-элемента-данных* определяет имя элемента **<data>** документа RFML, который во время выполнения программы содержит значение длины данных. *Имя-элемента-данных* можно задать только для элементов **<data>** с атрибутом **type=**`char` или **type=**`byte`. *Имя-элемента-данных* может быть задано полностью или относительно текущего элемента. В любом случае имя должно соответствовать элементу **<data** с атрибутом **type=**`int`. Дополнительная информация о преобразовании относительных имен приведена в разделе Преобразование относительных имен.	Задает длину элемента данных. Назначение этого атрибута зависит от типа данных. Дополнительная информация приведена в разделе Значения длины и точности.
name=	*имя*	Задает имя элемента **<data>**.
precision=	*число*	Задает число значимых разрядов для некоторых числовых типов данных. Дополнительная информация приведена в разделе Значения длины и точности.

Атрибут	Значение	Описание
struct=	*имя*	Задает имя элемента **<struct** в элементе **<data**. Атрибут **struct** можно задать только для элементов **<data>** с атрибутом **type=**`struct`.

Тег rfml языка RFML:

В теге rfml предусмотрены перечисленные ниже атрибуты. Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута.

```
<rfml version="версия"
    [ ccsid="число" ]>
</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).	Задает CCSID узла, которое применяется при работе со всеми вложенными элементами **<data со значением type**=″char″**>**, для которых не задан CCSID. Дополнительная информация приведена в описании тега **<data>** языка RFML. Если этот атрибут не задан, применяется CCSID хоста по умолчанию.

Тег recordformat языка RFML:

В теге recordformat предусмотрены перечисленные ниже атрибуты. Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута.

```
<recordformat name="имя"
    [ description="описание" ]>
</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:

В теге struct предусмотрены перечисленные ниже атрибуты. Атрибуты, заключенные в квадратные скобки [], являются необязательными. Скобки не являются частью атрибута.

```
<struct name="имя">
</struct>
```

Список атрибутов тега struct приведен в следующей таблице. Каждая запись содержит имя, допустимые значения и описание атрибута.

Атрибут	Значение	Описание
name=	*имя*	Задает имя элемента **<struct>**.

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. 14

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 
- Использовать любой совместимый анализатор XML в каталоге /QIBM/ProdData/OS400/xml/lib системы iSeries

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 обработчик XSLT Xerces с Web-сайта Apache 
- Использовать любой совместимый обработчик XSLT в каталоге /QIBM/ProdData/OS400/xml/lib системы iSeries

Эти обработчики можно запустить на сервере или скопировать их на рабочую станцию.

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
- Получать результаты работы программ iSeries в формате 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, в частности:

- Позволяет задавать и передавать значения параметров программ
- Позволяет получать результаты работы программ iSeries в формате 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

С помощью нового метода класса ProgramCallDocument, transformPCMLToXPCML, можно преобразовывать существующие документы PCML в эквивалентные документы XPCML. Это позволяет воспользоваться всеми преимуществами XPCML без создания отдельных исходных файлов XPCML для существующих документов вызова программ iSeries.

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

Компонент XPCML предъявляет те же требования к виртуальной машине Java рабочей станции, что и прочие компоненты IBM Toolbox for Java. Дополнительная информация приведена на следующей странице:

“Требования к рабочей станции для запуска приложений IBM Toolbox for Java” на стр. 11

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. В этом случае необходимо будет лишь задать расположение файла схемы с помощью атрибута xsi:noNamespaceSchemaLocation тега <xpcml>.

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. 417

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. 412

Схема и синтаксис XPCML

Документы XPCML, которые называются исходными файлами XPCML, содержат теги и данные, полностью определяющие вызовы программ в системе iSeries.

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 позволяет задавать начальное значение элемента данных в исходном файле PCML с помощью
| атрибута init тега <data>. 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

| Ниже приведены несколько простых сравнений, иллюстрирующих отличия XPCML от PCML. В каждом
| примере определяется вызов программы системы iSeries.

| Пример: Вызов программы сервера iSeries

| В следующем примере вызывается программа сервера iSeries с именем 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>
```

Пример: Вызов программы сервера iSeries с указанием массива параметров string

В следующем примере вызывается программа сервера iSeries с именем prog2 и в качестве параметра parm1 передается массив элементов типа string. 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:

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.

Для упрощения просмотра и печати документа некоторые строки этой версии HTML файла xpcml.xsd перенесены. Эти же строки в исходном файле .xsd не переносятся.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--////////////////////////////////////////////////////////////////////////
// 
// JTOpen (IBM Toolbox for Java - версия OSS)
// 
// Имя файла: xpcml.xsd
// 
// На исходный код в данном примере распространяется действие Общей лицензии IBM
// версии 1.0, утвержденной компанией Open Source Initiative.
// Copyright (C) 1997-2003 International Business Machines Corporation and
// others. Все права защищены.
// 
////////////////////////////////////////////////////////////////////////////-->

<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>

<xs:annotation>
 <xs:documentation>
  Схема для документов xpcml (расширенного языка вызова программ).
 </xs:documentation>
</xs:annotation>

<xs:element name="xpcml">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="structOrProgram" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="version" use="required">
     <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="4.0"/>
       </xs:restriction>
     </xs:simpleType>
   </xs:attribute>
  </xs:complexType>

  <!-- Определение ключа/ключевой связи между именем структуры и -->
  <!-- атрибутом struct поля параметра.        -->
  <xs:key name="StructKey">
    <xs:selector xpath="struct"/>
    <xs:field xpath="@name"/>
  </xs:key>
  <xs:keyref name="spRef" refer="StructKey">
     <xs:selector xpath="structParm" />
     <xs:field xpath="@struct" />
  </xs:keyref>
</xs:element>

<!-- Тег и атрибуты программы -->
<xs:element name="program" substitutionGroup="structOrProgram">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="parameterList" minOccurs="1" maxOccurs="1"/>
         <!-- Применяется в качестве тега-заменителя для списка параметров программы. -->
    </xs:sequence>
    <!-- Имя вызываемой программы. -->
    <xs:attribute name="name" type="string50" use="required" />
    <!-- Путь к объекту программы. По умолчанию она расположена в библиотеке QSYS. -->
    <xs:attribute name="path" type="xs:string"/>
    <!-- Указание порядка анализа параметров. -->
    <!-- Значение - это список имен параметров, разделенных пробелами. -->
    <xs:attribute name="parseOrder" type="xs:string"/>
    <!-- Имя точки входа служебной программы. -->
    <xs:attribute name="entryPoint" type="xs:string"/>
    <!-- Тип значения, если оно возвращается служебной программой. -->
    <xs:attribute name="returnValue" type="returnValueType"/>
```

```
|      <!-- Если при вызове программы на Java программа iSeries находится на том же сервере и -->
|      <!-- поддерживает нити, то этому параметру необходимо присвоить значение true, чтобы -->
|      <!-- программа iSeries вызывалась в том же задании и в той же нити, что и программа на Java. -->
|      <xs:attribute name="threadSafe" type="xs:boolean"  />
|      <!-- CCSID имени точки входа в служебной программе. -->
|      <xs:attribute name="epccsid" type="ccsidType"/>
|    </xs:complexType>
|  </xs:element>
|
|  <!-- Список состоит из одного или нескольких параметров. -->
|  <xs:element name="parameterList">
|    <xs:complexType>
|        <xs:group ref="programParameter" minOccurs="1" maxOccurs="unbounded"/>
|     </xs:complexType>
|  </xs:element>
|
|  <!-- Все поддерживаемые типы параметров программы. -->
|  <xs:group name="programParameter">
|        <xs:choice>
|   <xs:element ref="stringParmGroup"/>
|   <xs:element ref="stringParmArrayGroup"/>
|   <xs:element ref="intParmGroup"/>
|   <xs:element ref="intParmArrayGroup"/>
|          <xs:element ref="unsignedIntParmGroup"/>
|          <xs:element ref="unsignedIntParmArrayGroup"/>
|   <xs:element ref="shortParmGroup"/>
|   <xs:element ref="shortParmArrayGroup"/>
|   <xs:element ref="unsignedShortParmGroup"/>
|   <xs:element ref="unsignedShortParmArrayGroup"/>
|   <xs:element ref="longParmGroup"/>
|   <xs:element ref="longParmArrayGroup"/>
|   <xs:element ref="zonedDecimalParmGroup"/>
|   <xs:element ref="zonedDecimalParmArrayGroup"/>
|   <xs:element ref="packedDecimalParmGroup"/>
|          <xs:element ref="packedDecimalParmArrayGroup"/>
|   <xs:element ref="floatParmGroup"/>
|   <xs:element ref="floatParmArrayGroup"/>
|   <xs:element ref="doubleParmGroup"/>
|   <xs:element ref="doubleParmArrayGroup"/>
|   <xs:element ref="hexBinaryParmGroup"/>
|   <xs:element ref="hexBinaryParmArrayGroup"/>
|          <xs:element ref="structParmGroup"/>
|          <xs:element ref="structParmArrayGroup"/>
|          <xs:element ref="structArrayGroup"/>
|          <xs:element ref="struct"/>
|        </xs:choice>
|
|  </xs:group>
|
|  <!-- Абстрактный тип для всех типов параметров. -->
|  <xs:element name="stringParmGroup" type="stringParmType" abstract="true" />
|  <xs:element name="stringParmArrayGroup" type="stringParmArrayType" abstract="true" />
|  <xs:element name="intParmGroup" type="intParmType" abstract="true" />
|  <xs:element name="intParmArrayGroup" type="intParmArrayType" abstract="true" />
|  <xs:element name="unsignedIntParmGroup" type="unsignedIntParmType" abstract="true" />
|  <xs:element name="unsignedIntParmArrayGroup" type="unsignedIntParmArrayType" abstract="true" />
|  <xs:element name="shortParmGroup" type="shortParmType" abstract="true" />
|  <xs:element name="shortParmArrayGroup" type="shortParmArrayType" abstract="true" />
|  <xs:element name="unsignedShortParmGroup" type="unsignedShortParmType" abstract="true" />
|  <xs:element name="unsignedShortParmArrayGroup" type="unsignedShortParmArrayType" abstract="true" />
|  <xs:element name="longParmGroup" type="longParmType" abstract="true" />
|  <xs:element name="longParmArrayGroup" type="longParmArrayType" abstract="true" />
|  <xs:element name="zonedDecimalParmGroup" type="zonedDecimalParmType" abstract="true" />
|  <xs:element name="zonedDecimalParmArrayGroup" type="zonedDecimalParmArrayType" abstract="true" />
|  <xs:element name="packedDecimalParmGroup" type="packedDecimalParmType" abstract="true" />
|  <xs:element name="packedDecimalParmArrayGroup" type="packedDecimalParmArrayType" abstract="true" />
|  <xs:element name="floatParmGroup" type="floatParmType" abstract="true" />
```

```
| <xs:element name="floatParmArrayGroup" type="floatParmArrayType" abstract="true" />
| <xs:element name="doubleParmGroup" type="doubleParmType" abstract="true" />
| <xs:element name="doubleParmArrayGroup" type="doubleParmArrayType" abstract="true" />
| <xs:element name="hexBinaryParmGroup" type="hexBinaryParmType" abstract="true" />
| <xs:element name="hexBinaryParmArrayGroup" type="hexBinaryParmArrayType" abstract="true" />
| <xs:element name="structParmGroup" type="structParmType" abstract="true" />
| <xs:element name="structParmArrayGroup" type="structParmArrayType" abstract="true" />
| <xs:element name="structArrayGroup" type="structArrayType" abstract="true"
|             substitutionGroup="structOrProgram" />
|
| <!-- Параметр string -->
|    <xs:element name="stringParm" type="stringParmType" substitutionGroup="stringParmGroup"
|                nillable="true"/>
|      <xs:complexType name="stringParmType">
|        <xs:simpleContent>
|           <xs:extension base="stringFieldType">
|             <xs:attributeGroup ref="commonParmAttrs"/>
|           </xs:extension>
|        </xs:simpleContent>
|    </xs:complexType>
|
| <!-- Массив параметров string -->
|   <xs:element name="arrayOfStringParm" type="stringParmArrayType"
|               substitutionGroup="stringParmArrayGroup" nillable="true" />
|   <xs:complexType name="stringParmArrayType">
|     <xs:sequence>
|       <xs:element name="i" type="stringElementType" minOccurs="0" maxOccurs="unbounded"/>
|     </xs:sequence>
|     <xs:attributeGroup ref="commonParmAttrs"/>
|     <xs:attributeGroup ref="commonFieldAttrs"/>
|     <!-- Число элементов массива. -->
|     <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|     <xs:attribute name="count" type="xs:string" />
|     <!-- Число символов в каждом параметре string. -->
|     <xs:attribute name="length" type="xs:string"/>
|     <!-- CCSID хоста для каждого параметра string. -->
|     <xs:attribute name="ccsid" type="xs:string"/>
|     <!-- Задает способ усечения пробелов (слева, справа, слева и справа, не усекать). -->
|     <xs:attribute name="trim" type="trimType" />
|     <!-- Размер каждого символа (chartype в PCML). -->
|     <xs:attribute name="bytesPerChar" type="charType" />
|    <!-- Двунаправленные строки (string). -->
|     <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
|   </xs:complexType>
|
|       <xs:complexType name="stringElementType">
|         <xs:simpleContent>
|            <xs:extension base="xs:string">
|                 <!-- Указатель на элемент массива. -->
|                 <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|            </xs:extension>
|         </xs:simpleContent>
|        </xs:complexType>
|
|
| <!-- Параметр типа integer (4 байта на сервере) -->
|     <xs:element name="intParm" type="intParmType"  nillable="true" substitutionGroup="intParmGroup"   />
|       <xs:complexType name="intParmType" >
|          <xs:simpleContent>
|             <xs:extension base="intFieldType">
|               <xs:attributeGroup ref="commonParmAttrs"/>
|             </xs:extension>
|          </xs:simpleContent>
|       </xs:complexType>
|
| <!-- Массив элементов intParm -->
|   <xs:element name="arrayOfIntParm" type="intParmArrayType" substitutionGroup="intParmArrayGroup"
```

```
|                nillable="true" />
|    <xs:complexType name="intParmArrayType">
|      <xs:sequence>
|        <!-- i - это тег для указания элементов массива, не являющихся структурами. -->
|        <xs:element name="i" type="intElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="intElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:int">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|
|  <!-- Параметр integer без знака (4 байта на сервере) -->
|      <xs:element name="unsignedIntParm" type="unsignedIntParmType" nillable="true"
|                  substitutionGroup="unsignedIntParmGroup"   />
|        <xs:complexType name="unsignedIntParmType">
|           <xs:simpleContent>
|              <xs:extension base="unsignedIntFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив элементов intParm без знака -->
|    <xs:element name="arrayOfUnsignedIntParm" type="unsignedIntParmArrayType"
|                substitutionGroup="unsignedIntParmArrayGroup" nillable="true" />
|    <xs:complexType name="unsignedIntParmArrayType">
|      <xs:sequence>
|        <xs:element name="i" type="unsignedIntElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="unsignedIntElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:unsignedInt">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|
|  <!-- Параметр типа short (2 байта на сервере) -->
|     <xs:element name="shortParm" type="shortParmType" nillable="true" substitutionGroup="shortParmGroup"/>
|        <xs:complexType name="shortParmType">
|           <xs:simpleContent>
|              <xs:extension base="shortFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров shortParm -->
|    <xs:element name="arrayOfShortParm" type="shortParmArrayType" substitutionGroup="shortParmArrayGroup"
|                nillable="true" />
```

```
|    <xs:complexType name="shortParmArrayType">
|      <xs:sequence>
|        <xs:element name="i" type="shortElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="shortElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:short">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|
|  <!-- Параметр типа short без знака (2 байта на сервере) -->
|      <xs:element name="unsignedShortParm" type="unsignedShortParmType" nillable="true"
|                  substitutionGroup="unsignedShortParmGroup"   />
|        <xs:complexType name="unsignedShortParmType">
|           <xs:simpleContent>
|              <xs:extension base="unsignedShortFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров unsignedShortParm -->
|    <xs:element name="arrayOfUnsignedShortParm" type="unsignedShortParmArrayType"
|                substitutionGroup="unsignedShortParmArrayGroup" nillable="true" />
|    <xs:complexType name="unsignedShortParmArrayType">
|      <xs:sequence>
|        <xs:element name="i" type="unsignedShortElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="unsignedShortElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:unsignedShort">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|
|  <!-- Параметр типа long (8 байт на сервере) -->
|      <xs:element name="longParm" type="longParmType" nillable="true" substitutionGroup="longParmGroup"  />
|        <xs:complexType name="longParmType">
|           <xs:simpleContent>
|              <xs:extension base="longFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров longParm -->
|    <xs:element name="arrayOfLongParm" type="longParmArrayType" substitutionGroup="longParmArrayGroup"
|                nillable="true" />
|    <xs:complexType name="longParmArrayType">
|      <xs:sequence>
```

```
|        <xs:element name="i" type="longElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="longElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:long">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|
|  <!-- Параметр типа ZonedDecimal -->
|      <xs:element name="zonedDecimalParm" type="zonedDecimalParmType" nillable="true"
|                  substitutionGroup="zonedDecimalParmGroup"   />
|        <xs:complexType name="zonedDecimalParmType">
|           <xs:simpleContent>
|              <xs:extension base="zonedDecimalFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров zonedDecimalParm -->
|    <xs:element name="arrayOfZonedDecimalParm" type="zonedDecimalParmArrayType"
|                substitutionGroup="zonedDecimalParmArrayGroup" nillable="true" />
|    <xs:complexType name="zonedDecimalParmArrayType">
|      <xs:sequence>
|        <xs:element name="i" type="zonedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <!-- Общее число разрядов в поле (атрибут length в PCML). -->
|      <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
|      <!-- Число дробных разрядов (атрибут precision в PCML). -->
|      <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
|    </xs:complexType>
|
|    <xs:complexType name="zonedDecimalElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:decimal">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|  <!-- Параметр типа packedDecimal -->
|      <xs:element name="packedDecimalParm" type="packedDecimalParmType" nillable="true"
|                  substitutionGroup="packedDecimalParmGroup"   />
|        <xs:complexType name="packedDecimalParmType">
|           <xs:simpleContent>
|              <xs:extension base="packedDecimalFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров packedDecimalParm -->
|    <xs:element name="arrayOfPackedDecimalParm" type="packedDecimalParmArrayType"
|                substitutionGroup="packedDecimalParmArrayGroup" nillable="true" />
```

```
|    <xs:complexType name="packedDecimalParmArrayType">
|      <xs:sequence>
|        <xs:element name="i" type="packedDecimalElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attribute name="totalDigits" type="xs:positiveInteger"/>
|      <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger"/>
|    </xs:complexType>
|
|    <xs:complexType name="packedDecimalElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:decimal">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|  <!-- Параметр типа float (4 байта на сервере) -->
|     <xs:element name="floatParm" type="floatParmType" nillable="true" substitutionGroup="floatParmGroup"/>
|        <xs:complexType name="floatParmType">
|           <xs:simpleContent>
|              <xs:extension base="floatFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров floatParm -->
|    <xs:element name="arrayOfFloatParm" type="floatParmArrayType" substitutionGroup="floatParmArrayGroup"
|                nillable="true" />
|    <xs:complexType name="floatParmArrayType">
|      <xs:sequence>
|        <xs:element name="i" type="floatElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="floatElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:float">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|
|  <!-- Параметр типа double (8 байтов на сервере) -->
|      <xs:element name="doubleParm" type="doubleParmType" nillable="true"
|                  substitutionGroup="doubleParmGroup"   />
|        <xs:complexType name="doubleParmType">
|           <xs:simpleContent>
|              <xs:extension base="doubleFieldType">
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров doubleParm -->
|    <xs:element name="arrayOfDoubleParm" type="doubleParmArrayType"
|                substitutionGroup="doubleParmArrayGroup" nillable="true" />
|    <xs:complexType name="doubleParmArrayType">
```

```
|      <xs:sequence>
|        <xs:element name="i" type="doubleElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="doubleElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:double">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|
|  <!-- Двоичный параметр в шестнадцатеричном представлении (любое число байтов; без знака) -->
|      <xs:element name="hexBinaryParm" type="hexBinaryParmType"  substitutionGroup="hexBinaryParmGroup"  />
|        <xs:complexType name="hexBinaryParmType">
|           <xs:simpleContent>
|              <xs:extension base="hexBinaryFieldType">
|                <!-- Длина поля в байтах (атрибут length в PCML). -->
|                <xs:attribute name="totalBytes" type="xs:string"/>
|                <xs:attributeGroup ref="commonParmAttrs"/>
|              </xs:extension>
|           </xs:simpleContent>
|        </xs:complexType>
|
|  <!-- Массив параметров hexBinaryParm -->
|    <xs:element name="arrayOfHexBinaryParm" type="hexBinaryParmArrayType"
|                substitutionGroup="hexBinaryParmArrayGroup" nillable="true" />
|    <xs:complexType name="hexBinaryParmArrayType">
|      <xs:sequence>
|        <xs:element name="i" type="hexBinaryElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <xs:attribute name="totalBytes" type="xs:string"/>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|    </xs:complexType>
|
|    <xs:complexType name="hexBinaryElementType">
|      <xs:simpleContent>
|         <xs:extension base="xs:hexBinary">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:simpleContent>
|    </xs:complexType>
|
|  <!-- Параметр типа structure -->
|    <xs:element name="structParm" type="structParmType"  substitutionGroup="structParmGroup"   />
|        <xs:complexType name="structParmType">
|           <xs:complexContent>
|              <xs:extension base="structureParmArray">
|                <xs:attribute name="struct" type="string50"/>
|                <!-- Указывает, передается ли значение параметра или ссылка на него -->
|                <!-- (атрибут passby в PCML).-->
|                <!-- Значение допустимо только для целочисленных типов. -->
|                <xs:attribute name="passMode" type="passModeType"/>
|                <!-- Атрибут count необходим для создания массива входных/выходных -->
|                <!-- параметров в формате XPCML. -->
|                <xs:attribute name="count" type="xs:string"/>
|              </xs:extension>
|           </xs:complexContent>
```

```
|       </xs:complexType>
|
|  <!-- Массив параметров типа structure -->
|    <xs:element name="arrayOfStructParm" type="structParmArrayType"
|                substitutionGroup="structParmArrayGroup" nillable="true" />
|    <xs:complexType name="structParmArrayType">
|      <xs:sequence>
|          <!-- тег struct_i представляет отдельные параметры или элементы массива параметров struct. -->
|          <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <xs:attributeGroup ref="commonParmAttrs"/>
|      <xs:attributeGroup ref="commonFieldAttrs"/>
|      <xs:attribute name="struct" type="string50"/>
|    </xs:complexType>
|
|    <xs:complexType name="structElementType">
|      <xs:complexContent>
|         <xs:extension base="structureParmArray">
|              <xs:attribute name="index" type="xs:nonNegativeInteger"  />
|         </xs:extension>
|      </xs:complexContent>
|    </xs:complexType>
|
|
|  <!-- Элемент struct -->
|    <xs:element name="struct" type="structureParmArray" substitutionGroup="structOrProgram" />
|
|  <!-- Массив параметров struct -->
|    <xs:element name="arrayOfStruct" type="structArrayType" substitutionGroup="structArrayGroup"
|                nillable="true" />
|    <xs:complexType name="structArrayType">
|      <xs:sequence>
|         <!-- тег struct_i tag представляет элементы struct в массиве. -->
|         <xs:element name="struct_i" type="structElementType" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <!-- Имя элемента struct. -->
|      <xs:attribute name="name" type="string50"/>
|      <!-- Число элементов массива. -->
|      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
|      <xs:attribute name="count" type="xs:string" />
|      <!-- Указывает, для чего предназначен данный элемент struct:  -->
|      <!-- для ввода, для вывода или для ввода-вывода (атрибут usage в PCML). -->
|      <xs:attribute name="passDirection" type="passDirectionType"/>
|      <!-- Смещение элемента struct в выходном параметре. -->
|      <xs:attribute name="offset" type="xs:string" />
|      <!-- Базовое расположение для атрибута offset. -->
|      <xs:attribute name="offsetFrom" type="xs:string" />
|      <!-- Число байтов, которые следует резервировать для выходных данных элемента. -->
|      <xs:attribute name="outputSize" type="xs:string" />
|      <!-- Самая ранняя версия системы OS/400, поддерживающая данный элемент. -->
|      <xs:attribute name="minvrm" type="string10" />
|      <!-- Самая поздняя версия системы OS/400, поддерживающая данный элемент. -->
|      <xs:attribute name="maxvrm" type="string10" />
|    </xs:complexType>
|
|
|  <!-- Общие атрибуты для всех типов полей. -->
|    <xs:attributeGroup name="commonParmAttrs">
|       <!-- Указывает, для чего предназначен данный параметр:  -->
|       <!-- для ввода, для вывода или для ввода-вывода (атрибут usage в PCML). -->
|       <!-- Если значение не задано, по умолчанию применяется значение inherit. -->
|       <xs:attribute name="passDirection" type="passDirectionType"/>
|       <!-- Указывает, передается ли значение параметра или ссылка на него (атрибут passby в PCML).--> -->
|       <!-- Если значение не задано, по умолчанию применяется значение reference. -->
|       <xs:attribute name="passMode" type="passModeType" />
```

```
|       <!-- Смещение элемента в выходном параметре. -->
|       <!-- Если значение не задано, по умолчанию применяется значение 0. -->
|       <xs:attribute name="offset" type="xs:string" />
|       <!-- Базовое расположение для атрибута offset. -->
|       <xs:attribute name="offsetFrom" type="xs:string" />
|       <!-- Число байтов, которые следует резервировать для выходных данных элемента. -->
|       <xs:attribute name="outputSize" type="xs:string" />
|       <!-- Самая ранняя версия системы OS/400, поддерживающая данное поле. -->
|       <!-- Если значение не задано, считается, что поле поддерживается всеми версиями. -->
|       <xs:attribute name="minvrm" type="string10" />
|       <!-- Самая поздняя версия системы OS/400, поддерживающая данное поле. -->
|       <!-- Если значение не задано, считается, что поле поддерживается всеми версиями. -->
|       <xs:attribute name="maxvrm" type="string10" />
|    </xs:attributeGroup>
|
|    <xs:simpleType name="passDirectionType">
|      <xs:restriction base="xs:string">
|         <xs:enumeration value="in"/>
|         <xs:enumeration value="inout"/>
|         <xs:enumeration value="out"/>
|         <xs:enumeration value="inherit"/>
|      </xs:restriction>
|    </xs:simpleType>
|
|    <xs:simpleType name="passModeType">
|      <xs:restriction base="xs:string">
|         <xs:enumeration value="value"/>
|         <xs:enumeration value="reference"/>
|       </xs:restriction>
|    </xs:simpleType>
|
|    <!-- Следующие типы предназначены для обеспечения совместимости с PCML -->
|    <xs:simpleType name="bidiStringTypeType">
|      <xs:restriction base="xs:string">
|         <xs:enumeration value="ST4"/>
|         <xs:enumeration value="ST5"/>
|         <xs:enumeration value="ST6"/>
|         <xs:enumeration value="ST7"/>
|         <xs:enumeration value="ST8"/>
|         <xs:enumeration value="ST9"/>
|         <xs:enumeration value="ST10"/>
|         <xs:enumeration value="ST11"/>
|         <xs:enumeration value="DEFAULT"/>
|      </xs:restriction>
|    </xs:simpleType>
|
|    <xs:simpleType name="charType">
|      <xs:restriction base="xs:string">
|         <xs:enumeration value="onebyte"/>
|         <xs:enumeration value="twobyte"/>
|       </xs:restriction>
|    </xs:simpleType>
|
|    <xs:simpleType name="trimType">
|      <xs:restriction base="xs:string">
|         <xs:enumeration value="none"/>
|         <xs:enumeration value="left"/>
|         <xs:enumeration value="right"/>
|         <xs:enumeration value="both"/>
|       </xs:restriction>
|    </xs:simpleType>
|
|    <xs:simpleType name="returnValueType">
|      <xs:restriction base="xs:string">
|         <xs:enumeration value="void"/>
|         <xs:enumeration value="integer"/>
|       </xs:restriction>
```

```
|    </xs:simpleType>
|
|
|    <xs:complexType name="structureParmArray">
|      <xs:sequence>
|         <xs:group ref="structureParm" minOccurs="0" maxOccurs="unbounded"/>
|      </xs:sequence>
|      <xs:attribute name="name" type="string50"/>
|      <xs:attribute name="passDirection" type="passDirectionType"/>
|      <xs:attribute name="offset" type="xs:string" />
|      <xs:attribute name="offsetFrom" type="xs:string" />
|      <xs:attribute name="outputSize" type="xs:string" />
|      <xs:attribute name="minvrm" type="string10" />
|      <xs:attribute name="maxvrm" type="string10" />
|    </xs:complexType>
|
|
|  <!-- Параметр structureParm имеет один из следующих типов: stringParm, intParm,
|       shortParm, longParm, zonedDecimalParm, packedDecimalParm, floatParm,
|       doubleParm или hexBinaryParm. -->
|        <xs:group name="structureParm">
|          <xs:choice>
|            <xs:element ref="stringParmGroup"  />
|            <xs:element ref="stringParmArrayGroup"  />
|            <xs:element ref="intParmGroup"   />
|            <xs:element ref="intParmArrayGroup"   />
|            <xs:element ref="unsignedIntParmGroup"   />
|            <xs:element ref="unsignedIntParmArrayGroup"   />
|            <xs:element ref="shortParmGroup" />
|            <xs:element ref="shortParmArrayGroup" />
|            <xs:element ref="unsignedShortParmGroup"   />
|            <xs:element ref="unsignedShortParmArrayGroup"   />
|            <xs:element ref="longParmGroup"  />
|            <xs:element ref="longParmArrayGroup"  />
|            <xs:element ref="zonedDecimalParmGroup"  />
|            <xs:element ref="zonedDecimalParmArrayGroup"  />
|            <xs:element ref="packedDecimalParmGroup" />
|            <xs:element ref="packedDecimalParmArrayGroup" />
|            <xs:element ref="floatParmGroup" />
|            <xs:element ref="floatParmArrayGroup" />
|            <xs:element ref="doubleParmGroup" />
|            <xs:element ref="doubleParmArrayGroup" />
|            <xs:element ref="hexBinaryParmGroup" />
|            <xs:element ref="hexBinaryParmArrayGroup" />
|            <xs:element ref="structParmGroup" />
|            <xs:element ref="structParmArrayGroup"/>
|            <xs:element ref="structArrayGroup"/>
|            <xs:element ref="struct"/>
|          </xs:choice>
|        </xs:group>
|
|
|  <!-- Схема определений полей. -->
|
|    <!-- Определение основных исходных типов данных iSeries. -->
|
|       <xs:complexType name="zonedDecimal">
|          <xs:simpleContent>
|             <xs:extension base="xs:decimal">
|                  <xs:attribute name="totalDigits" type="xs:positiveInteger"  />
|                  <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger" />
|             </xs:extension>
|          </xs:simpleContent>
|       </xs:complexType>
|
|
|       <xs:complexType name="packedDecimal">
```

```
          <xs:simpleContent>
             <xs:extension base="xs:decimal">
                  <xs:attribute name="totalDigits" type="xs:positiveInteger" />
                  <xs:attribute name="fractionDigits" type="xs:nonNegativeInteger" />
             </xs:extension>
          </xs:simpleContent>
       </xs:complexType>


     <xs:complexType name="structureFieldArray">
      <xs:sequence>
         <xs:group ref="structureField" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="string50"/>
      <!-- Атрибут count необходим для создания массива входных/выходных параметров в формате XPCML. -->
      <xs:attribute name="count" type="xs:string"/>
    </xs:complexType>


    <!-- Абстрактный тип для типа struct or program. -->
    <xs:element name="structOrProgram" abstract="true" />


    <!-- Абстрактный тип для всех типов полей. -->
    <xs:element name="stringFieldGroup" type="stringFieldType" abstract="true" />
    <xs:element name="intFieldGroup" type="intFieldType" abstract="true" />
    <xs:element name="unsignedIntFieldGroup" type="unsignedIntFieldType" abstract="true" />
    <xs:element name="shortFieldGroup" type="shortFieldType" abstract="true" />
    <xs:element name="unsignedShortFieldGroup" type="unsignedShortFieldType" abstract="true" />
    <xs:element name="longFieldGroup" type="longFieldType" abstract="true" />
    <xs:element name="zonedDecimalFieldGroup" type="zonedDecimalFieldType" abstract="true" />
    <xs:element name="packedDecimalFieldGroup" type="packedDecimalFieldType" abstract="true" />
    <xs:element name="floatFieldGroup" type="floatFieldType" abstract="true" />
    <xs:element name="doubleFieldGroup" type="doubleFieldType" abstract="true" />
    <xs:element name="hexBinaryFieldGroup" type="hexBinaryFieldType" abstract="true" />
    <xs:element name="structFieldGroup" type="structFieldType" abstract="true" />


    <!-- Объявление каждого элемента поля отдельным типом поля. -->
    <xs:element name="stringField" type="stringFieldType" substitutionGroup="stringFieldGroup"
                nillable="true"/>
    <xs:element name="intField" type="intFieldType" nillable="true"
                substitutionGroup="intFieldGroup" />
    <xs:element name="unsignedIntField" type="unsignedIntFieldType"
                substitutionGroup="unsignedIntFieldGroup" nillable="true"/>
    <xs:element name="shortField" type="shortFieldType" nillable="true"
                substitutionGroup="shortFieldGroup" />
    <xs:element name="unsignedShortField" type="unsignedShortFieldType" nillable="true"
                substitutionGroup="unsignedShortFieldGroup"  />
    <xs:element name="longField" type="longFieldType" nillable="true"
                substitutionGroup="longFieldGroup" />
    <xs:element name="hexBinaryField" type="hexBinaryFieldType" nillable="true"
                substitutionGroup="hexBinaryFieldGroup" />
    <xs:element name="zonedDecimalField" type="zonedDecimalFieldType" nillable="true"
                substitutionGroup="zonedDecimalFieldGroup"  />
    <xs:element name="packedDecimalField" type="packedDecimalFieldType" nillable="true"
                substitutionGroup="packedDecimalFieldGroup" />
    <xs:element name="doubleField" type="doubleFieldType" nillable="true"
                substitutionGroup="doubleFieldGroup" />
    <xs:element name="floatField" type="floatFieldType" nillable="true"
                substitutionGroup="floatFieldGroup" />


    <xs:element name="structField" type="structFieldType" nillable="true"
                substitutionGroup="structFieldGroup" />
```

```
|
|  <!-- Параметр StructureField имеет один из следующих типов: stringField, intField,
|       shortField, longField, zonedDecimalField, packedDecimalField, floatField,
|       doubleField или hexBinaryField. -->
|        <xs:group name="structureField">
|          <xs:choice>
|            <xs:element ref="stringFieldGroup"/>
|            <xs:element ref="intFieldGroup"/>
|            <xs:element ref="unsignedIntFieldGroup"/>
|            <xs:element ref="shortFieldGroup"/>
|            <xs:element ref="unsignedShortFieldGroup"/>
|            <xs:element ref="longFieldGroup"/>
|            <xs:element ref="zonedDecimalFieldGroup"/>
|            <xs:element ref="packedDecimalFieldGroup"/>
|            <xs:element ref="floatFieldGroup"/>
|            <xs:element ref="doubleFieldGroup"/>
|            <xs:element ref="hexBinaryFieldGroup"/>
|            <xs:element ref="structParmGroup"/>
|            <xs:element ref="struct"/>
|          </xs:choice>
|        </xs:group>
|
|    <!-- Символьное поле -->
|    <!-- Соответствие типам объектов AS400Text. -->
|     <xs:complexType name="stringFieldType">
|       <xs:simpleContent>
|          <xs:extension base="xs:string">
|            <!-- Число символов. -->
|            <xs:attribute name="length" type="xs:string"/>
|            <!-- Указывает CCSID поля на сервере. -->
|            <xs:attribute name="ccsid" type="xs:string"/>
|            <xs:attribute name="trim" type="trimType" />
|            <xs:attribute name="bytesPerChar" type="charType" />
|            <xs:attribute name="bidiStringType" type="bidiStringTypeType" />
|            <xs:attributeGroup ref="commonFieldAttrs"/>
|          </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|
|    <!-- Поле hexBinary -->
|    <!-- Преобразуется в объект AS400ByteArray. -->
|     <xs:complexType name="hexBinaryFieldType">
|       <xs:simpleContent>
|          <xs:extension base="xs:hexBinary">
|            <xs:attributeGroup ref="commonFieldAttrs"/>
|          </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|
|    <!-- Поле float -->
|    <!-- Преобразуется в объект AS400Float4. -->
|     <xs:complexType name="floatFieldType">
|       <xs:simpleContent>
|           <xs:extension base="xs:float">
|               <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|      </xs:complexType>
|
|    <!-- Поле zonedDecimal -->
|    <!-- Преобразуется в объект AS400ZonedDecimal. -->
|     <xs:complexType name="zonedDecimalFieldType">
|       <xs:simpleContent>
|           <xs:extension base="zonedDecimal">
|               <xs:attributeGroup ref="commonFieldAttrs"/>
```

```
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|    <!-- Поле packedDecimal -->
|    <!-- Преобразуется в объект AS400PackedDecimal. -->
|     <xs:complexType name="packedDecimalFieldType">
|       <xs:simpleContent>
|           <!-- В DDS значения binary имеют от 1 до 18 разрядов; если длина поля превосходит
|                9 символов, то десятичные разряды должны быть равны 0. -->
|           <xs:extension base="packedDecimal">
|               <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|    <!-- Поле int -->
|    <!-- Преобразуется в объект AS400Bin4. -->
|     <xs:complexType name="intFieldType">
|       <xs:simpleContent>
|           <xs:extension base="xs:int">
|              <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|    <!-- Поле int без знака -->
|    <!-- Преобразуется в объект AS400Bin4. -->
|     <xs:complexType name="unsignedIntFieldType">
|       <xs:simpleContent>
|           <xs:extension base="xs:unsignedInt">
|              <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|    <!-- Поле short -->
|    <!-- Преобразуется в объект AS400Bin2. -->
|     <xs:complexType name="shortFieldType">
|       <xs:simpleContent>
|           <xs:extension base="xs:short">
|              <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|    <!-- Поле short без знака -->
|    <!-- Преобразуется в объект AS400Bin2. -->
|     <xs:complexType name="unsignedShortFieldType">
|       <xs:simpleContent>
|           <xs:extension base="xs:unsignedShort">
|              <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|
|    <!-- Поле long -->
|    <!-- Преобразуется в объект AS400Bin8. -->
|     <xs:complexType name="longFieldType">
|       <xs:simpleContent>
|           <xs:extension base="xs:long">
|              <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
```

```
|    <!-- Поле double -->
|    <!-- Преобразуется в объект AS400Float8. -->
|     <xs:complexType name="doubleFieldType">
|       <xs:simpleContent>
|           <xs:extension base="xs:double">
|              <xs:attributeGroup ref="commonFieldAttrs"/>
|           </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|
|
|    <!-- Поле struct -->
|     <xs:complexType name="structFieldType">
|       <xs:simpleContent>
|         <xs:extension base="xs:string">
|           <xs:attribute name="struct" type="string50"/>
|           <xs:attributeGroup ref="commonFieldAttrs"/>
|         </xs:extension>
|       </xs:simpleContent>
|     </xs:complexType>
|
|
|  <!-- Общие атрибуты для всех типов полей. -->
|  <xs:attributeGroup name="commonFieldAttrs">
|     <xs:attribute name="name" type="string50"/>
|     <xs:attribute name="columnHeading1" type="string20"/>
|     <xs:attribute name="columnHeading2" type="string20"/>
|     <xs:attribute name="columnHeading3" type="string20"/>
|     <xs:attribute name="description" type="string50"/>
|     <xs:attribute name="defaultValue" type="xs:string"/><!-- Max length of string is 65535 characters. -->
|     <xs:attribute name="nullable" type="xs:boolean"/>
|     <xs:attribute name="isEmptyString" type="xs:boolean"/><!-- Indicate this is an empty string. -->
|  </xs:attributeGroup>
|
|  <!-- Служебные типы.  -->
|
|  <xs:simpleType name="ccsidType">
|                       <xs:restriction base="xs:nonNegativeInteger">
|                        <xs:maxInclusive value="65535"/>
|                       </xs:restriction>
|                      </xs:simpleType>
|
|  <xs:simpleType name="string10">
|                       <xs:restriction base="xs:string">
|                        <xs:maxLength value="10"/>
|                       </xs:restriction>
|                      </xs:simpleType>
|
|  <xs:simpleType name="string20">
|                       <xs:restriction base="xs:string">
|                        <xs:maxLength value="20"/>
|                       </xs:restriction>
|                      </xs:simpleType>
|
|  <xs:simpleType name="string50">
|                       <xs:restriction base="xs:string">
|                        <xs:maxLength value="50"/>
|                       </xs:restriction>
|                      </xs:simpleType>
|  </xs:schema>
```

| **Синтаксис XPCML:**

| В схеме XPCML определяются несколько тегов элементов, каждый из которых 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	
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	

Тег XPCML	Описание	Эквивалентный тег PCML
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	

Тег XPCML	Атрибут	Описание
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)
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

- Получение результатов работы программ iSeries в формате 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
Содержит информацию о преобразовании существующих документов PCML в эквивалентные документы XPCML.

Вызов программ на сервере iSeries с помощью XPCML
Эта страница содержит сведения о создании объектов ProgramCallDocument, которые с помощью XPCML вызывают программы на сервере.

Получение результатов вызова программы в формате XPCML
Этот раздел содержит информацию о том, как можно получить результаты вызова программы в формате XPCML.

Передача значений параметров в формате XPCML
Этот раздел содержит информацию об указании значений параметров в формате XPCML и их передаче в программы. Он содержит описания нескольких способов указания постоянных значений параметров и массивов данных.

Применение уплотненного XPCML
В этом разделе описано уплотнение документов XPCML, упрощающее их чтение и применение. Кроме того, в нем описано создание объектов ProgramCallDocument с помощью документов XPCML и получение результатов вызова программ в виде уплотненных документов XPCML.

Идентификация ошибок анализатора в формате XPCML
В этом разделе приведена информация об идентификации и занесении в протокол различных предупреждений и устранимых ошибок анализатора, которые могут возникать при анализе документа XPCML.

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="Значение 1"/>
   </program>
</pcml>
```

Код Java, преобразующий файл myPCML.pcml в 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">Значение 1
         </stringParm>
      </parameterList>
   </program>
</xpcml>
```

Дополнительная информация о методе transformPCMLToXPCML() и классе ProgramCallDocument приведена на следующей странице:

Справочная информация Java для класса ProgramCallDocument

Вызов программ на сервере iSeries с помощью XPCML:

После создания файла XPCML необходимо создать объект ProgramCallDocument, который с помощью спецификаций и значений данных этого файла будет вызывать программу в системе iSeries. 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. 414

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.

Вызов программ на сервере iSeries с помощью XPCML

После того, как объект ProgramCallDocument создан, для работы с документом XPCML можно применять любые методы класса ProgramCallDocument. Например, можно вызвать программу в системе iSeries с помощью метода ProgramCallDocument.callProgram() или изменить значение входного параметра XPCML перед вызовом программы на сервере с помощью соответствующего метода ProgramCallDocument.setValue.

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.

Дополнительная информация о классе ProgramCallDocument приведена на следующей странице:

Справочная информация Java для класса ProgramCallDocument

В приведенном ниже примере создается объект XPCML ProgramCallDocument и вызывается программа iSeries, результаты которой возвращаются в виде документа XPCML.

“Пример: Получение результатов вызова программы в виде файла XPCML” на стр. 741

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. 743

“Пример: Передача массивов значений параметров в виде файла XPCML” на стр. 745

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. 415

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. 749

“Пример: Получение результатов вызова программы в виде уплотненного файла XPCML” на стр. 750

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)

Дополнительная информация о методе condenseXPCML() и классе ProgramCallDocument приведена на следующей странице:

Справочная информация Java для класса ProgramCallDocument

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. 746

“Пример: Уплотнение существующего документа XPCML с исходным кодом Java” на стр. 747

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)

К часто задаваемым вопросам (FAQ) относятся вопросы, связанные с повышением производительности IBM Toolbox for Java, устранением неполадок, применением JDBC и т.д.:

- IBM Toolbox for Java FAQ : содержит ответы на различные вопросы, в том числе о повышении производительности, работе с OS/400, устранении неполадок и многие другие.
- IBM Toolbox for Java JDBC FAQ : Здесь вы найдете ответы на вопросы о применении JDBC с IBM Toolbox for Java

Советы программисту

В этом разделе приведен ряд советов по работе с IBM Toolbox for Java:

Завершение работы программы на Java
Информация о том, как правильно завершать работу программы на Java.

Применение путей интегрированных файловых систем
Информация о применении путей интегрированных файловых систем в программах. В этом разделе описаны формат путей в интегрированных файловых системах, а также параметры и специальные значения.

Управление соединениями
Инструкции по установлению и завершению соединения с сокетом с помощью класса AS400, в том числе согласно спецификации Enterprise JavaBean.

Работа с виртуальной машиной Java (JVM) OS/400
Информация о работе с классами IBM Toolbox for Java в JVM OS/400. Этот раздел содержит рекомендации по оптимизации доступа к ресурсам сервера, по работе с классами и по настройке паролей при входе в систему.

Подключение к независимому ASP (IASP)
Указания по подключению к IASP. Независимый пул вспомогательной памяти (IASP) - это набор дисков, который можно включить или выключить независимо от остальной памяти системы.

Обработка ошибок при работе с классами доступа Toolbox for Java
Описание способов обработки ошибок с помощью классов исключительных ситуаций IBM Toolbox for Java в программах, применяющих классы доступа IBM Toolbox for Java.

Обработка ошибок при работе с классами графического интерфейса IBM Toolbox for Java
Описание способов обработки ошибок с помощью классов событий при ошибках IBM Toolbox for Java в программах, применяющих классы графического интерфейса IBM Toolbox for Java.

Работа с классом Trace
Указания по применению класса Trace для занесения точек трассировки и диагностических сообщений в протокол при отладке программ.

Оптимизация программ
Информация о способах оптимизации программ с целью повышения их производительности.

Повышение производительности за счет применения JVM OS/400
Информация о повышении производительности за счет применения JVM OS/400.

Управление классами IBM Toolbox for Java в клиентской системе
Указания по работе с классами IBM Toolbox for Java на клиенте с помощью класса AS400ToolboxInstaller.

Повышение производительности при работе с JAR-файлами
В этом разделе описано создание маленьких, быстро загружающихся JAR-файлов IBM Toolbox for Java с помощью класса JarMaker IBM Toolbox for Java.

Поддержка национальных языков в Java
Информация о поддержке национальных языков в IBM Toolbox for Java.

Обслуживание и поддержка
Здесь приведена информация о службах поддержки IBM Toolbox for Java.

Завершение работы программы на Java

Для корректного завершения работы программы 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 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). Имя в интегрированной файловой системе - это имя объекта сервера в том формате, в котором оно применяется в библиотечной файловой системе интегрированной файловой системы сервера iSeries.

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

Различные классы IBM Toolbox for Java поддерживают специальные значения в именах интегрированной файловой системы. Обычно в командах iSeries такие значение начинаются со звездочки (***ALL**). Тем не менее, в программе на Java, в которой применяются классы IBM Toolbox for Java, эти специальные значения начинаются и заканчиваются знаками процентов (**%ALL%**).

Примечание: В интегрированной файловой системе символом подстановки является звездочка.

В приведенной ниже таблице указано, какие специальные значения поддерживаются классами IBM Toolbox for Java в различных элементах пути. В таблице также приведены различия между стандартным форматом этих значений и форматом, применяемым в классах IBM Toolbox for Java.

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.

Управление соединениями

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.

Для подключения к системе iSeries в программе на Java нужно создать объект AS400. Объект AS400 содержит по одному соединению с сокетом для каждого типа сервера iSeries. Каждая служба является заданием сервера iSeries и предоставляет доступ к данным этого сервера.

Примечание: При создании объектов Enterprise JavaBeans (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.

Для каждого соединения с сервером в системе iSeries создается отдельное задание. Большинство серверов поддерживают следующие службы:

- JDBC
- Вызов программ и команд
- Интегрированная файловая система
- Печать
- Очередь данных
- Доступ на уровне записей

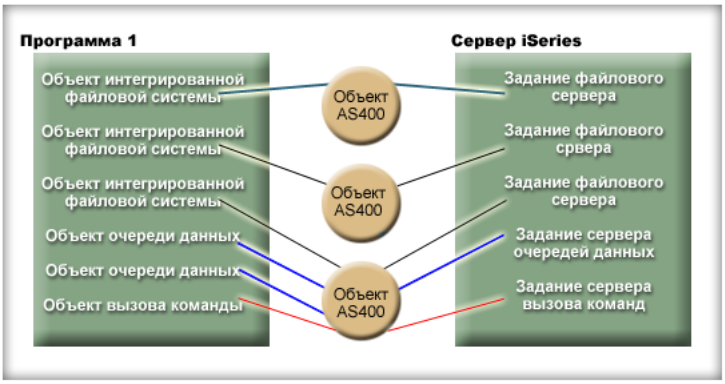
Примечание:

- 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.

Программа на Java может изменять число соединений с сервером iSeries. Для оптимизации работы соединений программа на Java может создать несколько объектов AS400 для одной системы, как показано на рис. 1 (для системы iSeries устанавливается несколько соединений с сокетами).

Рисунок 1: Создание в программе на Java несколько объектов AS400 и соединений с сокетами одного сервера iSeries



Для минимизации объема ресурсов сервера iSeries, затрачиваемого на обмен данными, создайте только один объект AS400, как показано на рис. 2. Это позволяет сократить число соединений и объем занятых ресурсов системы.

Рисунок 2: Создание одного объекта AS400 и одного соединения с сокетами iSeries в программе на Java

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.

Кроме того, вы можете управлять соединениями, создав пул соединений, как показано на рис. 3. Такой подход позволяет сократить время, затрачиваемое на подключение к iSeries, за счет повторного применения ранее установленных соединений.

Рисунок 3: Применение в программе на Java соединения с сервером iSeries из пула AS400ConnectionPool

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();
```

Пример 2: В этом примере создаются два объекта CommandCall, которые вызывают команды в одной системе iSeries. 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. Поскольку объекты CommandCall и IFSFileInputStream обращаются к разным службам системы iSeries, будет создано два соединения.

```
        // 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();
```

Пример 4: В приведенном ниже примере соединение с iSeries получено из пула AS400ConnectionPool. 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.

Ниже приведены примеры программ на Java, которые подключаются и отключаются от iSeries.

Пример 1: В этом примере показано, как можно заранее подключиться к серверу iSeries:

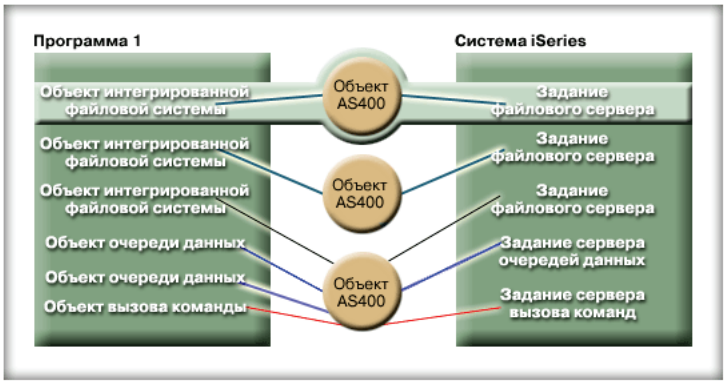
```
        // Создание объекта AS400.
AS400 system1 = new AS400("mySystem.myCompany.com");
```

```
        // 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 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. Впоследствии
        // 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. Впоследствии
        // она не понадобится.
sys.disconnectService(AS400.FILE);
```

Подробное описание рисунка 1: программа на Java, создающая несколько объектов AS400 и соединений между сокетами для одной системы iSeries (rzahh549.gif)

IBM Toolbox for Java: Управление соединениями

Этот рисунок иллюстрирует создание программой на Java нескольких объектов AS400, с каждым из которых связано отдельное соединение с системой iSeries. Создание нескольких соединений увеличивает объем используемых системой ресурсов.

Описание

Рисунок состоит из следующих компонентов:

- Прямоугольника слева, представляющего программу на Java
- Прямоугольника справа, представляющего программу на Java
- Трех небольших кругов, выровненных по вертикали между этими прямоугольниками и представляющих объекты AS400, созданные программой на Java для подключения к системе iSeries

Программа на Java (прямоугольник слева) содержит объекты нескольких типов, использующие объекты AS400 (три небольших круга). Объекты программы связаны с объектами AS400 линиями. Каждому объекту программы требуется для работы только один объект AS400. В то же время, один объект AS400 может применяться для подключения к системе iSeries несколькими объектами программы.

Линии соединяют объекты AS400 со службами системы iSeries (прямоугольник справа). Эти службы соответственно дублируют объекты программы. Связи нескольких объектов программы, трех объектов AS400 и служб системы iSeries перечислены в следующем списке:

- Объект интегрированной файловой системы в программе связан с первым объектом AS400, подключенным к заданию файлового сервера
- Другой объект интегрированной файловой системы в программе связан со вторым объектом AS400, подключенным ко второму заданию файлового сервера
- С третьим объектом AS400 связаны несколько объектов программы: третий объект интегрированной файловой системы, два объекта очереди данных и объект вызова команды. С другой стороны этот объект AS400 связан со следующими службами системы iSeries: третьим заданием файлового сервера (для объекта интегрированной файловой системы), заданием сервера очередей данных (для объектов очереди данных) и заданием сервера вызова команд (для объекта вызова команды).

Подробное описание рисунка 2: Программа на Java, создающая один объект AS400 и одно соединение с сокетами для системы iSeries (rzahh552.gif)

IBM Toolbox for Java: Управление соединениями

Данный рисунок иллюстрирует, как работа нескольких объектов программы с одним объектом AS400 уменьшает число соединений и объем используемых ресурсов системы.

Описание

Рисунок состоит из следующих компонентов:

- Прямоугольника слева, представляющего программу на Java
- Прямоугольника справа, представляющего программу на Java
- Небольшого круга между этими прямоугольниками, представляющего единственный объект AS400, созданный программой на Java для подключения к системе iSeries

Программа на Java (прямоугольник слева) содержит объекты нескольких типов, использующие объект AS400. Объекты программы связаны с объектом AS400 (небольшой круг) линиями.

Линии соединяют объекты AS400 со службами системы iSeries (прямоугольник справа). Каждая служба требует отдельного соединения с сервером. Эти службы соответственно дублируют объекты программы:

- Объект очереди данных в программе обращается к объекту AS400 для подключения к заданию сервера очередей данных
- Два объекта интегрированной файловой системы обращаются к объекту AS400 для подключения к заданию файлового сервера
- Объект вызова команды обращается к объекту AS400 для подключения к заданию вызова команд.

Все эти объекты программы работают с одним объектом AS400, создающим отдельные соединения с каждым заданием сервера. В этом случае, поскольку объекты интегрированной файловой системы работают с одним объектом AS400, они совместно используют одно соединение с заданием сервера.

Подробное описание рисунка 3: Программа на Java, применяющая соединение с сервером iSeries из пула AS400ConnectionPool (rzahh506.gif)

IBM Toolbox for Java: Управление соединениями

Данный рисунок иллюстрирует применение пула соединений для управления соединениями сервера iSeries.

Описание

Рисунок состоит из следующих компонентов:

- Прямоугольника слева, представляющего приложение на Java
- Изображения справа сервера iSeries, представляющего сервер, с которым программа на Java пытается установить соединение
- Овала между изображениями слева и справа, представляющего объект AS400ConnectionPool

Внутри объекта AS400ConnectionPool (в форме овала) показаны соединения. Каждое соединение - это объект AS400 в виде небольшого изображения сервера iSeries с рисунком молнии на нем:

- Программа на Java (прямоугольник слева) и пул соединений (овал) соединены круговой стрелкой.
- Сервер iSeries (изображение справа) соединен с пулом соединений (овал) простой линией.

Верхняя половина круговой стрелки указывает от объекта AS400ConnectionPool (овал) на приложение Java (прямоугольник слева). На стрелке показано одно из соединений пула, а одно из изображений соединения в овале пула показано нечетко. Это означает, что соединение из пула было запрошено и используется приложением на Java.

Нижняя половина круговой стрелки указывает от приложения на Java на пул соединений. Эта стрелка обозначает возврат приложением на Java соединения в пул.

Простая линия, соединяющая сервер iSeries (изображение справа) с объектом AS400ConnectionPool (овал), представляет соединения через сокеты с сервером.

Подробное описание рисунка 4: Объект программы работает с собственным объектом AS400 и экземпляром службы. Соединение прерывается. (rzahh550.gif)

IBM Toolbox for Java: Управление соединениями

На этом рисунке показано, что отключение одного объекта AS400 не влияет на соединения других объектов AS400.

Описание

Рисунок состоит из тех же элементов, что и рисунок 1:

- Прямоугольника слева, представляющего программу на Java
- Прямоугольника справа, представляющего программу на Java
- Трех небольших кругов, выровненных по вертикали между этими прямоугольниками и представляющих объекты AS400, созданные программой на Java для подключения к системе iSeries

Программа на Java (прямоугольник слева) содержит объекты нескольких типов, использующие объекты AS400 (три небольших круга). Объекты программы связаны с объектами AS400 линиями. Каждому объекту программы требуется для работы только один объект AS400. В то же время, один объект AS400 может применяться для подключения к системе iSeries несколькими объектами программы.

Линии соединяют объекты AS400 со службами системы iSeries (прямоугольник справа). Эти службы соответственно дублируют объекты программы. Информация о различных объектах программы, объектах AS400 и службах сервера iSeries приведена в описании Рисунка 1.

Соединение через один из объектов AS400 выделено, демонстрируя, что отключение этого соединения не повлияет на соединения других объектов AS400. Соединение состоит из объекта интегрированной файловой системы, используемого объекта AS400 и связанной службы системы iSeries (задания файлового сервера) и представлено связывающей их линией. Другие объекты программы, AS400 и соединения не будут затронуты.

Подробное описание рисунка 5: Все объекты программы работают с одним объектом AS400 и экземпляром службы. Соединение прерывается. (rzahh551.gif)

IBM Toolbox for Java: Управление соединениями

Этот рисунок иллюстрирует, что отключение соединения со службой одного объекта AS400 приводит к отключению всех объектов программы, использующих этот объект AS400 и работающих с данной службой.

Описание

Рисунок состоит из тех же элементов, что и рисунок 2:

- Прямоугольника слева, представляющего программу на Java
- Прямоугольника справа, представляющего программу на Java
- Небольшого круга между этими прямоугольниками, представляющего единственный объект AS400, созданный программой на Java для подключения к системе iSeries

Программа на Java (прямоугольник слева) содержит объекты нескольких типов, использующие объект AS400 (небольшой круг). Объекты программы связаны с объектом AS400 линиями.

Линии соединяют объекты AS400 со службами системы iSeries (прямоугольник справа). Эти службы соответственно дублируют объекты программы. Информация о различных объектах программы, объектах AS400 и службах сервера iSeries приведена в описании Рисунка 2.

Два объекта интегрированной файловой системы обращаются к объекту AS400 для подключения к заданию файлового сервера системы iSeries. Два объекта программы, связанная с ними служба и соединяющие их линии выделены. Отключение службы повлияет только на выделенные элементы, то есть приведет к отключению обоих объектов программы. Другие объекты программы, службы и соединения, а также сам объект AS400 не будут затронуты.

Виртуальная машина Java для OS/400

Классы IBM Toolbox for Java выполняются на виртуальной машине Java продукта IBM Developer Kit for Java (OS/400). В действительности классы могут применяться на любой платформе, поддерживающей спецификации Java 2 Software Development Kit (J2SDK).

Если для работы с классами IBM Toolbox for Java применяется JVM для OS/400, выполните следующие действия:

- Укажите, какое средство должно применяться для доступа к ресурсам сервера iSeries при работе с виртуальной машиной Java для OS/400: JVM для OS/400 или классы IBM Toolbox for Java.
- Ознакомьтесь с разделом Применение классов IBM Toolbox for Java в JVM для OS/400.
- Ознакомьтесь с информацией о том, как задать имя системы, ИД пользователя и пароль в JVM для OS/400.

Дополнительная информация о поддержке различных платформ Java на сервере iSeries приведена в разделе Поддержка различных JDK.

Сравнение Виртуальной машины Java для OS/400 и классов IBM Toolbox for Java

Программа, выполняемая виртуальной машиной Java для IBM Developer Kit for Java (OS/400), может обращаться к ресурсам системы с помощью одного из следующих интерфейсов:

- Встроенные функции Java
- Классы 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:
  - Программа запускается только на клиенте
  - Программа запускается только на сервере
  - Программа запускается на клиенте и на сервере, но обращается только к ресурсам сервера iSeries
  - Программа запускается в JVM для OS/400 и обращается к ресурсам другого сервера iSeries
  - Программа запускается на серверах различных типов

  Если программа запускается на клиенте и на сервере (включая случай, когда одна система iSeries выступает в роли клиента для другой системы iSeries), и при этом обращается только к ресурсам сервера iSeries, то рекомендуется использовать интерфейс IBM Toolbox for Java.

  Если программа обращается к серверам различных типов, то рекомендуется использовать стандартные интерфейсы Java.
- **Совместимость / Переносимость** - Если в системе iSeries установлены классы IBM Toolbox for Java, то в программах клиента и сервера могут применяться одни и те же интерфейсы. В этом случае вам не потребуется изучать два набора интерфейсов, что ускорит вашу работу.

  Однако программы, в которых применяются интерфейсы IBM Toolbox for Java, будут поддерживаться только одним типом **серверов**.

  Если программа будет применяться как в системе iSeries, так и на других серверах, рекомендуется использовать стандартные методы Java.

- **Сложность** - Интерфейс IBM Toolbox for Java специально создан для упрощения доступа к ресурсам сервера iSeries. Чаще всего вместо классов IBM Toolbox for Java вам придется создавать собственные программы, обращающиеся к ресурсам и взаимодействующие с основной программой через Стандартный интерфейс Java (JNI).

  Решите, что лучше: создавать собственные программы, применяющие стандартный интерфейс Java, или 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. Однако программа с классом IFSFileOutputStream может применяться только на серверах iSeries. При применении класса IBM Toolbox for Java вы не сможете использовать **сервер**.

  Решите, что важнее: переносимость или возможность пользоваться дополнительными средствами.
- **Ресурсы** - Если программа выполняется Виртуальной машиной Java для OS/400, 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.
- **iSeries в роли клиента** - Если программа запущена в одной системе iSeries, а обращается к ресурсам другой системы iSeries, то рекомендуется применять классы IBM Toolbox for Java. Они предоставляют простой интерфейс для доступа к ресурсам другой системы iSeries.

  Примером может служить очередь данных. Интерфейсы очереди данных лицензионной программы IBM Toolbox for Java обеспечивают доступ к очереди данных системы AS/400.

  Классы IBM Toolbox for Java позволяют обратиться к очереди данных сервера iSeries как с клиента, так и с сервера. Кроме того, они могут применяться для создания программы, которая запускается на одной системе iSeries, а обращается к очереди данных другой системы iSeries.

  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, созданная таким способом, будет переносимой, если вы создадите одну версию основной программы и несколько версий методов для обращения к очередям данных серверов различных типов.

Выполнение классов IBM Toolbox for Java в Виртуальной машине Java для OS/400

Ниже приведены некоторые рекомендации по выполнению классов IBM Toolbox for Java на Виртуальной машине Java (JVM), входящей в состав IBM Developer Kit for Java (OS/400):

Вызов команд

Существует два стандартных способа вызова команд:

- С помощью класса CommandCall IBM Toolbox for Java
- 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

Существует несколько способов обращения к файлу интегрированной файловой системы сервера iSeries:

- С помощью класса IFSFile лицензионной программы IBM Toolbox for Java
- С помощью классов файлов из пакета java.io

Классы интегрированной файловой системы IBM Toolbox for Java поддерживают больше функций, чем классы java.io. Классы IBM Toolbox for Java можно применять в аплетах. Кроме того, для получения информации с сервера им не требуется функция для перенаправления вывода (например, iSeries Access для Windows).

Преимуществом классов java.io является то, что они поддерживаются многими платформами. Их рекомендуется применять в тех программах, которые будут обращаться к серверам различных типов.

Если на компьютере-клиенте применяются классы java.io, то для работы с файлами сервера вам потребуется функция для перенаправления вывода (например, iSeries Access для Windows).

JDBC

Для программ, выполняемых виртуальной машиной Java для OS/400, предусмотрены следующие драйверы JDBC фирмы IBM:

- Драйвер JDBC для IBM Toolbox for Java
- Драйвер JDBC для IBM Developer Kit for Java

Если программы выполняются в среде клиент-сервер, то рекомендуется применять драйвер JDBC IBM Toolbox for Java.

Если программы выполняются на сервере iSeries, то рекомендуется применять драйвер JDBC для IBM Developer Kit for Java.

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:

- С помощью класса ProgramCall из набора IBM Toolbox for Java
- Посредством Стандартного интерфейса Java (JNI)

Класс ProgramCall лицензионной программы IBM Toolbox for Java обладает тем преимуществом, что он может вызывать любые программы сервера iSeries.

JNI позволяет вызывать лишь некоторые программы сервера iSeries. Однако JNI поддерживается большим числом платформ.

Настройка имени системы, ИД пользователя и пароля с помощью объекта AS400 в Виртуальной машине Java для OS/400

Если программа на Java выполняется на виртуальной машине (JVM) IBM Developer Kit for Java (OS/400), объект AS400 позволяет указать системные значения для имени системы, ИД пользователя и пароля.

Ниже приведена информация о некоторых специальных значениях, а также другие рекомендации по выполнению программ в Виртуальной машине Java для OS/400:

- 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 обращении к отдельным записям файла в системе версии v4r3 или ниже необходимо задать пароль. 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.
- Специальное значение ***current** можно указать в качестве ИД пользователя или пароля в объекте AS400 в том случае, если программа на Java выполняется в виртуальной машине Java для OS/400, и при этом обращается к ресурсам другой системы iSeries. 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

Ниже приведены примеры применения объекта AS400 в программе, выполняемой в JVM OS/400.

Пример: Создание объекта AS400 при выполнении в JVM OS/400 программы на Java

Если программа на Java выполняется в JVM OS/400, она может не предоставлять системное имя, ИД пользователя и пароль.

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.

Если программа выполняется виртуальной машиной Java для OS/400, то имя системы **localhost** эквивалентно пустому имени. Ниже приведен пример подключения к текущему серверу:

```
// Создание двух объектов AS400. Если программа на Java выполняется
// в JVM для OS/400, 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")
```

Пример: Подключение к текущему серверу с иными ИД пользователя и паролем, нежели в задании, запустившем программу В программе на Java могут быть заданы ИД пользователя и пароль, даже если она работает в JVM OS/400. 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")
```

Пример: Подключение к другому серверу с ИД пользователя и паролем задания, запустившего программу на Java

Программа на Java, запущенная в системе, может подключаться и использовать ресурсы других серверов iSeries.

Если в качестве ИД пользователя и пароля будет задано значение ***current**, то при подключении к целевой системе iSeries будут применяться ИД пользователя и пароль задания, запустившего программу на Java.

В приведенном ниже примере программа, 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 программами на Java на сервере и 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	**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.

Во всех остальных классах IBM Toolbox for Java (IFSFile, Print, DataQueues и др.) применяется IASP, указанный в описании задания пользовательского профайла, под управлением которого установлено соединение с сервером. Конец изменений

Оптимизация OS/400

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.

Вместе с OS/400 поставляются некоторые дополнительные классы, которые оптимизируют работу IBM Toolbox for Java в виртуальной машине Java для iSeries. Если программа выполняется виртуальной машиной Java для iSeries и подключается к локальной системе iSeries, то ее производительность повышается, а время входа в систему уменьшается. Дополнительные классы поставляются вместе с OS/400, начиная с выпуска V4R3.

Включение функции оптимизации

IBM Toolbox for Java поставляется в виде двух пакетов: в качестве отдельной лицензионной программы и вместе с OS/400.

- Лицензионная программа 5722-JC1. Файлы лицензионной программы IBM Toolbox for Java расположены в следующем каталоге:

  ```
  /QIBM/ProdData/http/public/jt400/lib
  ```

  Эти файлы не содержат классы, применяемые для оптимизации программ в OS/400. Эти файлы IBM Toolbox for Java следует применять в том случае, если программа на сервере и на клиенте должна выполняться с одинаковой скоростью.
- OS/400. IBM Toolbox for Java поставляется вместе с OS/400 в каталоге

  ```
  /QIBM/ProdData/OS400/jt400/lib
  ```

  Эти файлы содержат классы, применяемые для оптимизации выполнения программ, созданных с помощью IBM Toolbox for Java, в виртуальной машине Java для iSeries.

  Дополнительная информация приведена в Примечании 1 к разделу Файлы Jar.

Соглашения о входе в систему

Дополнительные классы OS/400 позволяют программе на Java задать имя системы, ИД пользователя и пароль для IBM Toolbox for Java.

При обращении к ресурсам iSeries имя системы, ИД пользователя и пароль должны быть предоставлены классом IBM Toolbox for Java.

- **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 у пользователя при входе в систему.
- **Если программа выполняется в виртуальной машине Java для iSeries**, появляется еще одна полезная возможность. В этом случае программа на Java может отправлять запросы локальному серверу, указывая ИД пользователя и пароль запустившего ее задания.

Дополнительные классы дают возможность применять ИД пользователя и пароль текущего задания и для подключения программы на Java к другой системе iSeries. 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.

```
                    // Создание объекта AS400. Используется конструктор
                    // по умолчанию, в котором не задается имя системы,
                    // ИД пользователя и пароль. Следовательно, объект AS400
                    // будет отправлять запросы локальной системе iSeries
                    // с помощью ИД пользователя и пароля задания. Если бы
                    // программа запускалась на клиенте, то появлялось бы
                    // приглашение на ввод имени системы, ИД пользователя и пароля.
AS400 sys1 = new AS400();

                    // Создание объекта AS400. Объект отправляет
                    // запросы в локальную систему iSeries с помощью
                    // ИД пользователя и пароля задания. Такой вариант
                    // инициализации неприменим в клиентской системе.
AS400 sys2 = new AS400("localhost", "*current", "*current");

                    // Создание двух объектов вызова команд, работающих
                    // с ранее созданными объектами AS400.
CommandCall cmd1 = new CommandCall(sys1,"myCommand1");
CommandCall cmd2 = new CommandCall(sys2,"myCommand2");

                    // Запуск команд.
cmd1.run();
cmd2.run();
```

Пример: Вход в систему с помощью ИД пользователя и пароля текущего задания

В следующем примере создается объект AS400, представляющий вторую систему iSeries. Поскольку задано значение ″*current″, для подключения ко второй (целевой) системе iSeries будет применяться ИД пользователя и пароль задания, запустившего программу на Java в исходной системе iSeries.

```
                    // Создание объекта AS400. Объект отправляет
                    // запросы второй системе iSeries с помощью ИД пользователя
                    // 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

Дополнительные классы OS/400 позволяют повысить производительность программ, выполняемых виртуальной машиной Java для iSeries. Повышение производительности связано с возможностью сокращения числа вызовов функций связи и применения API iSeries вместо обращения к серверам.

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

Все функции IBM Toolbox for Java, за исключением функций для работы с JDBC и интегрированной файловой системой, выполняются виртуальной машиной Java для iSeries быстрее. Это связано с тем, виртуальная машина Java вызывает меньше функций связи для подключения программы на Java к программе сервера iSeries.

Функции для работы с JDBC и интегрированной файловой системы не были оптимизированы, так как соответствующие средства их оптимизации уже существуют. При запуске программы в системе iSeries рекомендуется применять драйвер JDBC для iSeries вместо драйвера JDBC, поставляемого вместе с IBM Toolbox for Java. Для работы с файлами сервера рекомендуется использовать пакет java.io вместо классов доступа к интегрированной файловой системе, входящих в состав IBM Toolbox for Java.

Прямой вызов API iSeries

Следующие классы, входящие в состав IBM Toolbox for Java, обеспечивают высокую производительность, поскольку в них вместо вызова программы сервера напрямую вызываются API iSeries.

- 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. Система iSeries выбирала номер свободного порта и возвращала его пользователю. 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. Это связано с тем, что система iSeries просматривает сообщения в файле класса быстрее, чем в файле свойств. 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

Повышение эффективности преобразования данных Java в данные iSeries обеспечивается двумя классами:

- Программа преобразования двоичных данных: Преобразует массивы байт Java в простые типы данных Java.
- Программа преобразования символьных данных: Позволяет преобразовывать строковые объекты Java в кодовые страницы OS/400 и наоборот.

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 языка 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, IBM Toolbox for Java пытается преобразовать данные с помощью средств языка Java. Таблицы преобразования больше не загружаются с сервера. 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.

Рекомендации по оптимизации программы с помощью команды Создать программу на Java (CRTJVAPGM)

Если приложение на Java работает в JVM системы iSeries, вы можете **значительно повысить ее производительность**, создав программу на Java с помощью файла .zip или .jar IBM Toolbox for Java. Для этого введите в командной строке iSeries: **CRTJVAPGM**. (Дополнительная информация о команде **CRTJVAPGM** приведена в электронной справке по этой команде.) Команда **CRTJVAPGM** позволяет сохранить созданную программу на Java, содержащую классы IBM Toolbox for Java, при ее запуске. Сохранение программы на Java позволяет значительно сократить время запуска. Такая экономия достигается за счет того, что программу не приходится создавать заново при каждом запуске.

В версиях V4R2 и V4R3 продукта IBM Toolbox for Java команда **CRTJVAPGM** не поддерживает файлы jt400.zip и jt400.jar, так как их размер слишком велик. Однако эту команду можно запустить для файла jt400Access.zip. В версии V4R3 лицензионная программа IBM Toolbox for Java содержит дополнительный файл, jt400Access.zip. Этот файл содержит только классы доступа. В нем нет визуальных классов.

При работе с приложениями на Java в системе версии V4R5 (или более ранней), следует применять файл jt400Access.zip. В системе версии V5R1, следует применять файл jt400Native.jar. Файл jt400Access.zip уже обработан командой **CRTJVAPGM**.

Поддержка национальных языков в Java

Java поддерживает не все национальные языки, предусмотренные на сервере.

Если применяется неподдерживаемый язык, например, при работе на локальной рабочей станции, язык которой не поддерживается Java, лицензионная программа IBM Toolbox for Java **может выдавать сообщения об ошибках на английском языке**.

Обслуживание и поддержка IBM Toolbox for Java

Ниже перечислены различные источники информации и средства, применяемые для обслуживания и поддержки:

Сведения об устранении неполадок IBM Toolbox for Java Этот раздел содержит информацию, которая поможет вам устранить неполадки в работе продукта IBM Toolbox for Java.

Конференция JTOpen/IBM Toolbox for Java На этой странице вы сможете обменяться информацией с разработчиками на Java, применяющими IBM Toolbox for Java. На этом форуме вам с удовольствием окажут помощь программисты на Java, а может быть - даже сами разработчики Toolbox for Java.

Поддержка сервера На Web-сайте поддержки сервера IBM Server приведены сведения о средствах и ресурсах, которые помогут вам обеспечить непрерывное планирование и поддержку сервера iSeries.

Поддержка программного обеспечения Web-сайт IBM Software Support Services содержит информацию о самых разных службах поддержки программного обеспечения IBM.

Услуги по поддержке продукта IBM Toolbox for Java, 5722-JC1, оказываются на тех же условиях, что и поддержка других продуктов для iSeries. Услуги по поддержке включают программные службы, консультации специалистов и 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:

- Вопросы, связанные с ошибками API Java. Вы можете проверить, связана ли ошибка с AS/400 Toolbox for Java, 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. Дополнительные примеры программ можно найти в Internet на Домашней странице iSeries . Эти примеры не поддерживаются.

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

Ниже перечислены ссылки на точки входа примеров программ, приведенных в разделе IBM Toolbox for Java.

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	ToolboxMe for iSeries	Классы утилит
Классы Vaccess	Классы XPCML	

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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: Классы доступа

В этом разделе перечислены примеры программ, встречающиеся в документации по классам доступа IBM Toolbox for Java.

AS400JPing

- Пример: Применение AS400JPing в программе на Java

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:

> **Отказ от гарантий на предоставляемый код**
>
> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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

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.
//
// Пример применения класса CommandCall в IBM Toolbox for Java
//
////////////////////////////////////////////////////////////////////////////



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 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

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

```
////////////////////////////////////////////////////////////////////////////////
//
// Пример работы с классом AS400ConnectionPooling. Эта программа использует
// AS400ConnectionPool для создания соединений с сервером iSeries.
// 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();
      }
   }
}
```

Пример: Применение классов FieldDescription, RecordFormat и Record

Ниже приведены примеры применения классов FieldDescription, RecordFormat и Record с очередями данных.

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

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,
// числа преобразуются из формата Java в формат сервера. Преобразованные
// записи очереди данных могут быть считаны программой сервера,
// программой iSeries Access for Windows или другой программой на Java.
//
// 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.
             // Он совпадает с форматом в классе DQConsumer.
             // 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("Введите число ==>");
         }
      }

      return i;
   }
}
```

Пример: Применение классов DataQueue для считывания записей из очереди данных

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.
             // Он совпадает с форматом в классе DQProducer.
             // 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

Вы можете использовать классы AS400DataType с ProgramCall для передачи параметров в программы и интерпретации полученных параметров.

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.
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)

Получаемое значение	Тип
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

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("Введите число ==>");
         }
      }

      return i;
   }
}
```

Пример: Применение классов KeyedDataQueue для считывания записей из очереди данных

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 со средним 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

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

Ниже приведены примеры применения класса IFSFile:

- 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. 489

Пример: Создание каталога

```
                        // Создание объекта AS400. Новый
                        // каталог будет создан в этой
                        // системе iSeries.
      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

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++)
            {
                // Печать списка.
                // 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

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

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

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);
    }



}
```

Пример: Применение класса JDBCQuery для отправки запроса к таблице

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

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;
   }

                // Присвоение значений переменным.
                // 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

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

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("Формат:  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

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)               // Установка очереди сообщений
    {
        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

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)

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)

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

```
///////////////////////////////////////////////////////////////////////////
//
// В данном примере список буферных файлов системы создается асинхронно
// 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

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

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
   // является выходным. Выделение 64 байтов для этого параметра.

   parmlist[0] = new ProgramParameter( 64 );



   // Параметр 2 задает размер буфера для параметра 1. Он является входным
   // параметром. 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. 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. 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

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

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

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
//
// Эта программа копирует записи из файла-примера базы данных iSeries Access
// 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
  // будет неупорядоченным, размер блока должен быть равен одной записи.
  // Параметр фиксации уровня блокировки игнорируется, поскольку не было
  // 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

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: JavaBean

В этом разделе перечислены примеры программ, встречающиеся в документации по компонентам JavaBean IBM Toolbox for Java.

- Пример: Использование прослушивающих функций для вывода комментариев при подключении и отключении от системы и запуске команд
- Пример: Использование аплетов и IBM VisualAge для Java для создания кнопок и запуска команд

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код примеров

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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
```

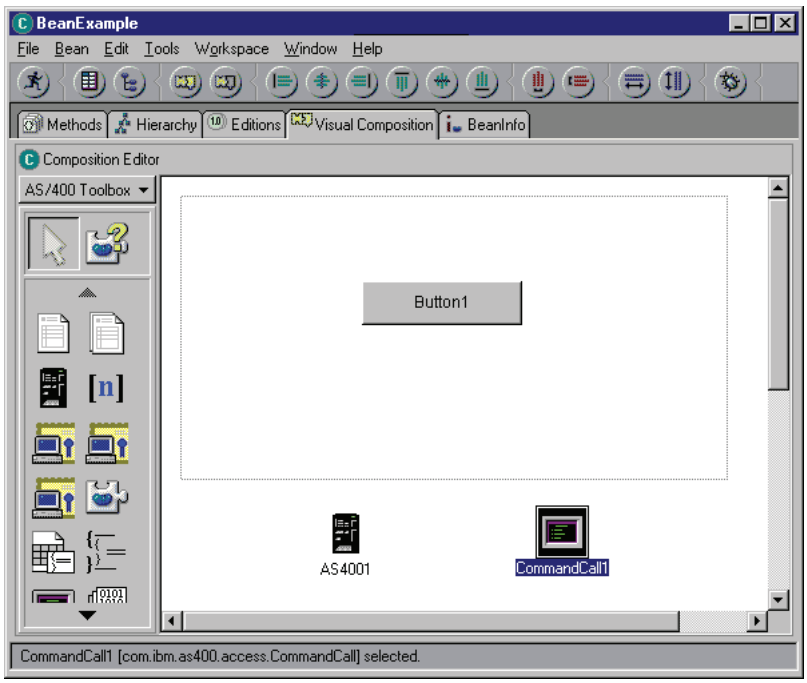
```
        // с помощью обработчика событий объекта commandCall.
        // Объект будет вызывать этот код при каждом запуске команды.

        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. В примере создается аплет, формирующий кнопку, при нажатии которой запускается команда на сервере iSeries.

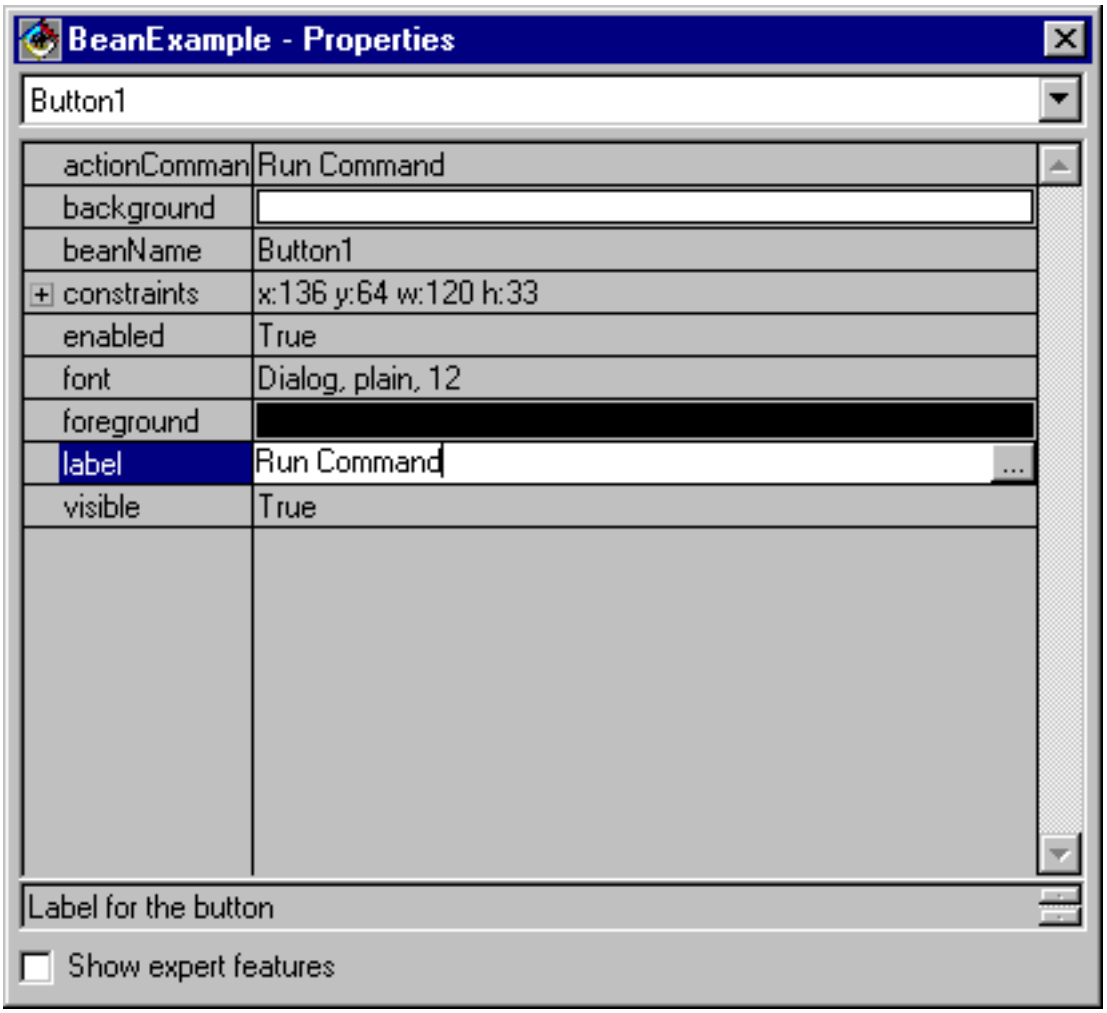
- Перенесите кнопку в окно аплета. (Кнопка находится в левой части окна визуального компоновщика, как показано на рисунке 1.)
- Перенесите компоненты CommandCall и AS400 за пределы окна аплета. (Компоненты находятся в левой части окна визуального компоновщика, как показано на рисунке 1.)

Рисунок 1: Окно визуального компоновщика VisualAge - gui.BeanExample



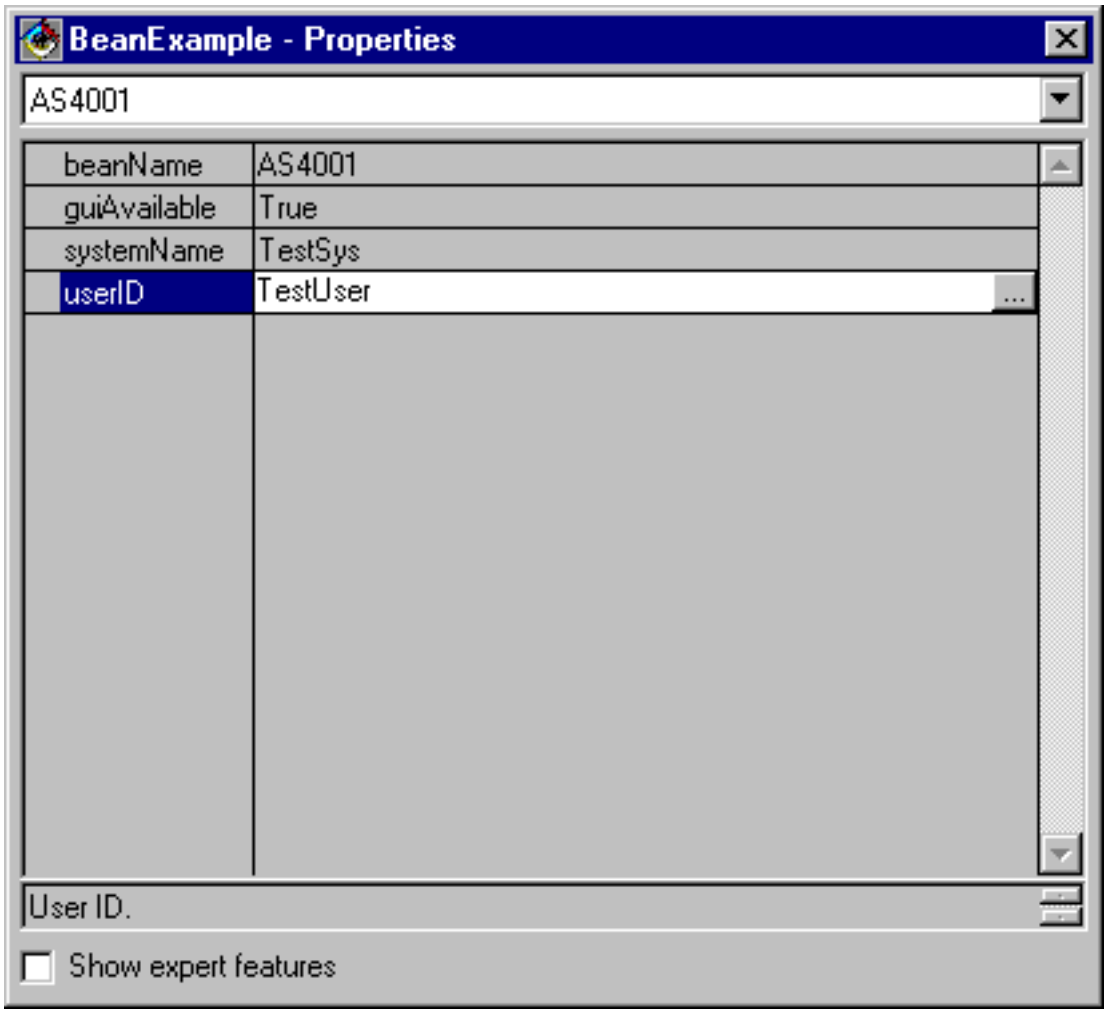
- Отредактируйте свойства компонента. (Выберите компонент и нажмите правую кнопку мыши, затем в выпадающем окне выберите опцию Свойства.)
  - Измените название кнопки на **Запустить команду**, как показано на рисунке 2.

    Рисунок 2: Изменение названия кнопки на "Запустить команду"



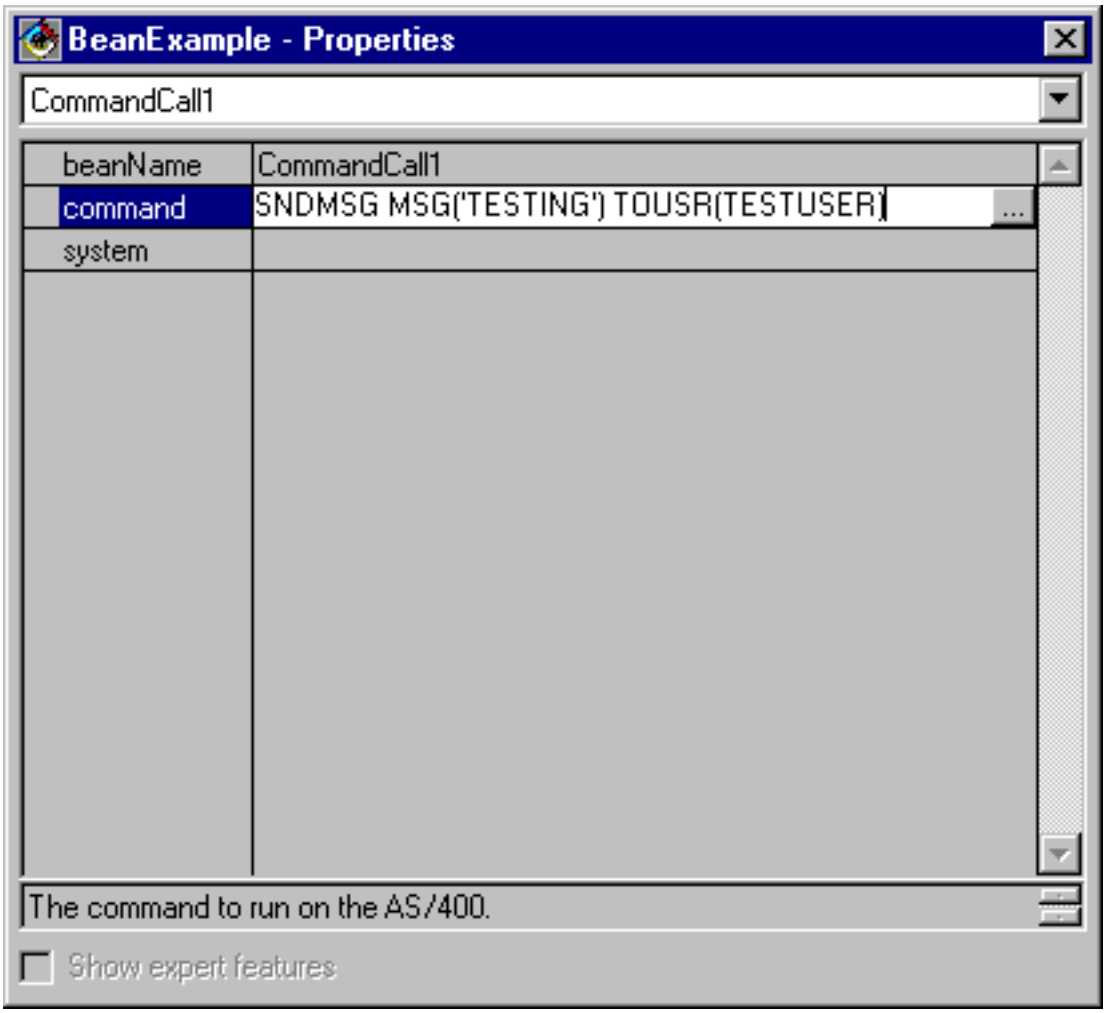
– Измените имя системы в компоненте AS400 на **TestSys**
– Измените ИД пользователя в компоненте AS400 на **TestUser**, как показано на рисунке 3.

Рисунок 3: Изменение ИД пользователя на TestUser



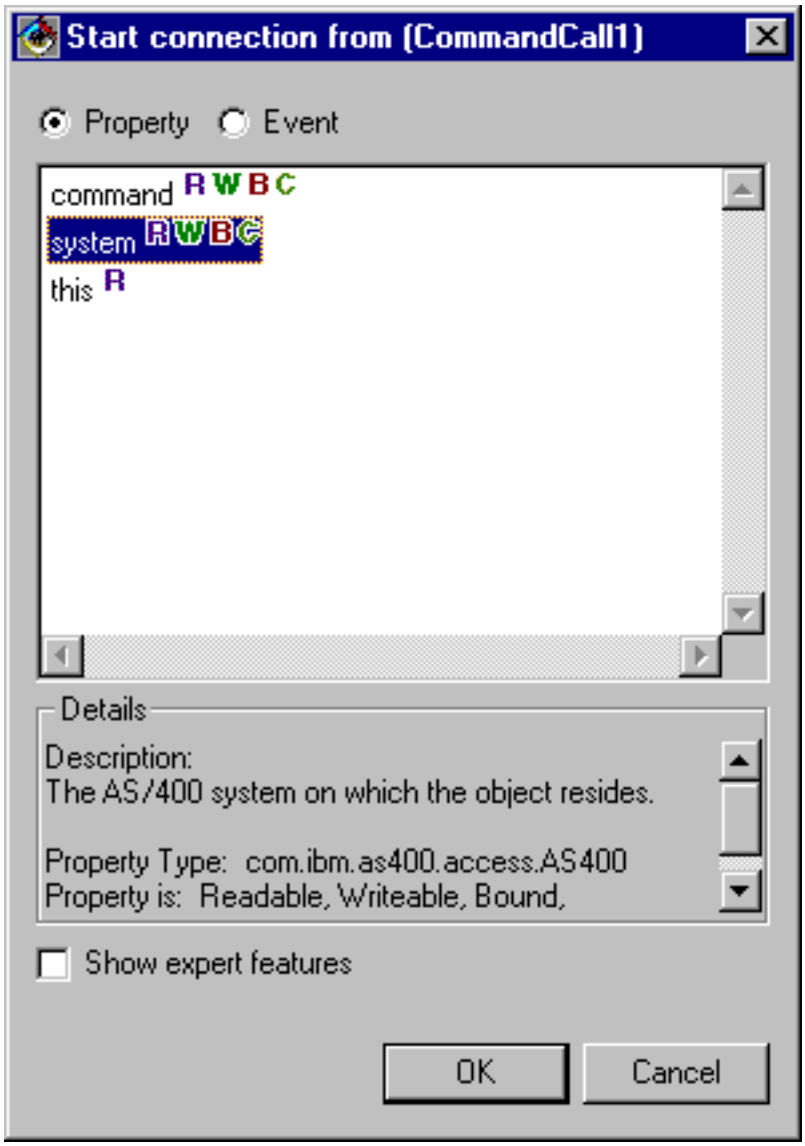
– Измените текст команды в компоненте CommandCall на **SNDMSG MSG('Проверка') TOUSR('TESTUSER')**, как показано на рисунке 4.

Рисунок 4: Изменение команды в компоненте CommandCall



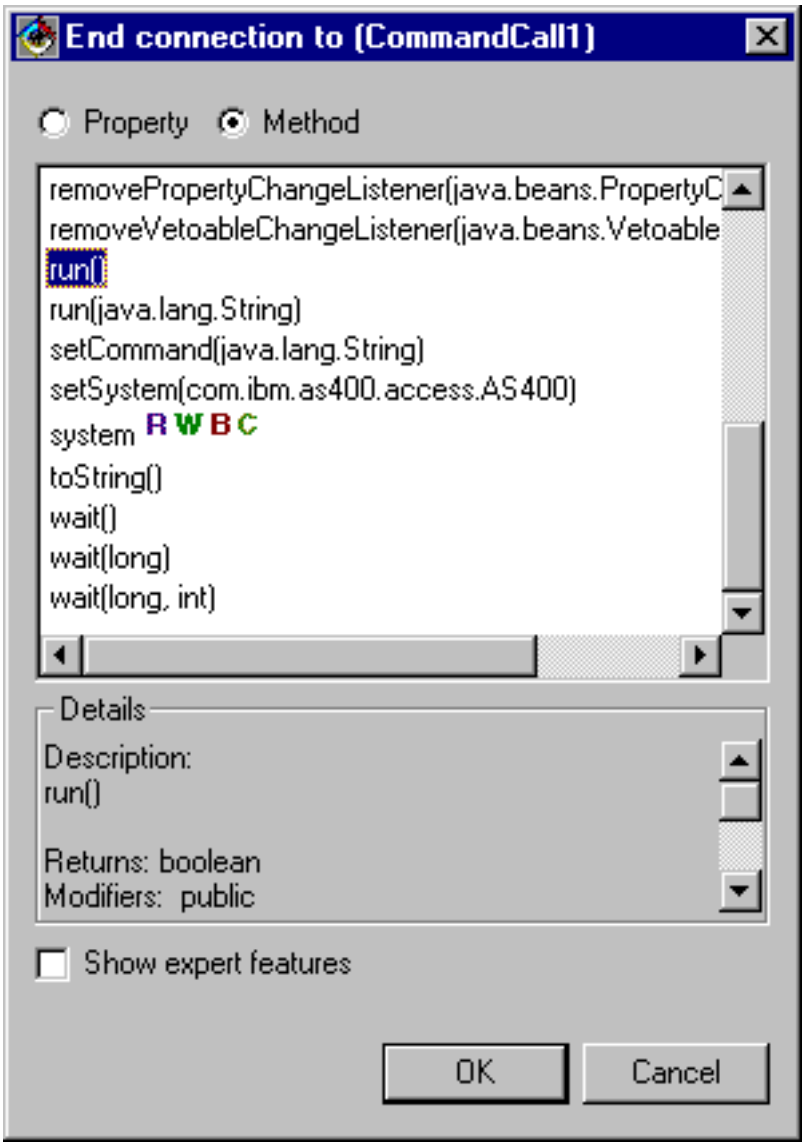
- Свяжите компонент AS400 с компонентом CommandCall. Способ связывания может быть различным в разных компоновщиках. В данном примере сделайте следующее:
  - Выберите компонент CommandCall и нажмите правую кнопку мыши
  - Выберите **Связать**
  - Выберите **Опции связывания**
  - Выберите **system** в 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



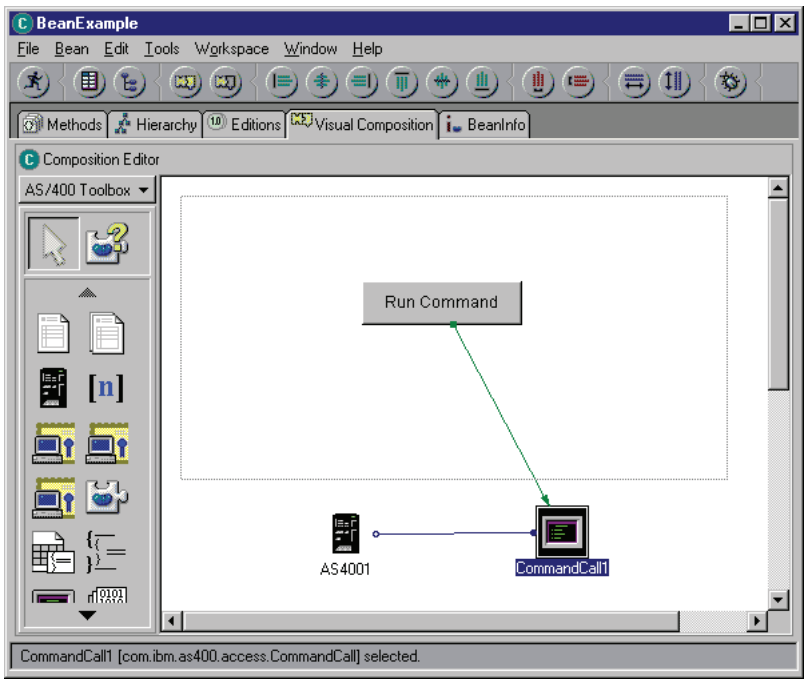
- Свяжите кнопку с компонентом CommandCall.
  - Выберите компонент Кнопка и нажмите правую кнопку мыши
  - Выберите **Связать**
  - Выберите **actionPerformed**
  - Выберите компонент CommandCall
  - Выберите **Connectable Features** из появившегося выпадающего меню
  - Выберите **run()** в списке методов, 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

Эта страница содержит ссылки на примеры программ, приведенные в документации по классам трассировки соединений IBM Toolbox for Java.

- “Пример: Применение классов Commtrace” на стр. 186

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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:

> **Отказ от гарантий на предоставляемый код**
>
> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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: Создание панели с помощью GUI Builder

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

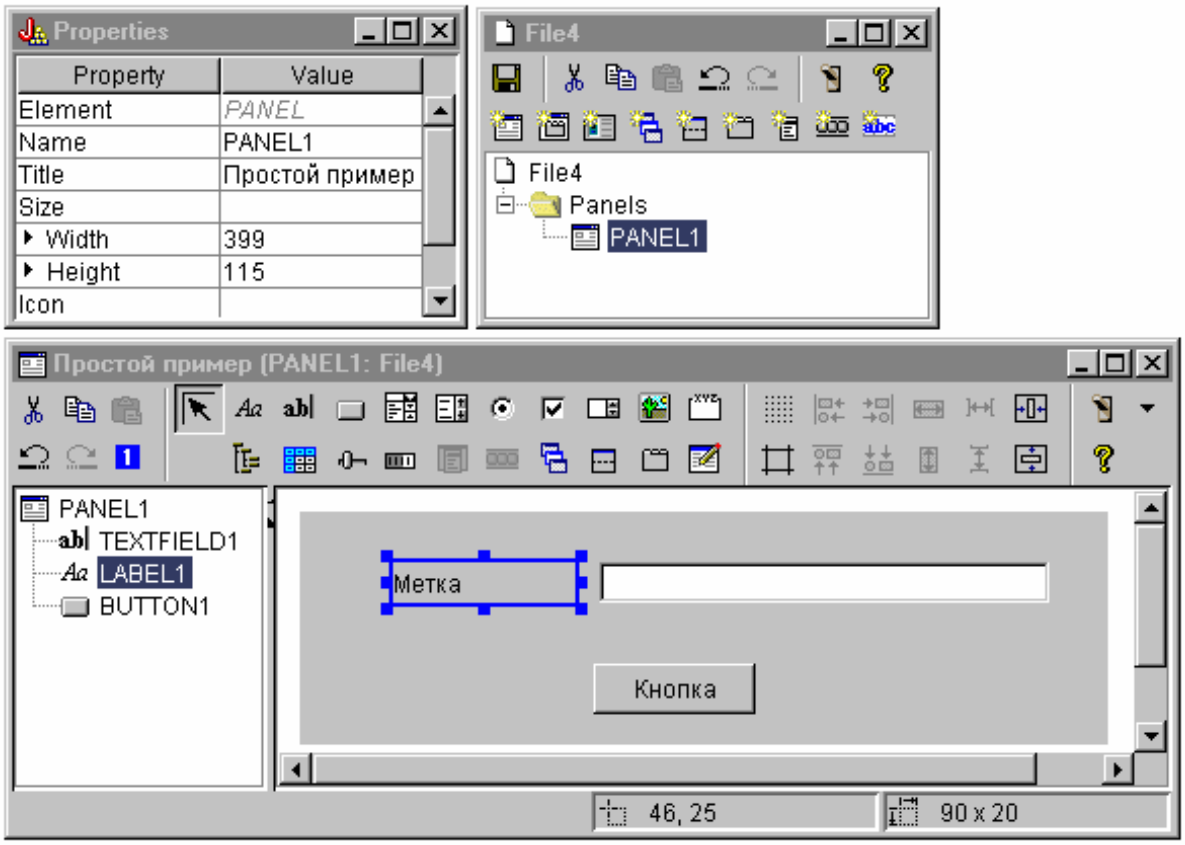
Ниже приведен пример создания простой панели с помощью набора Graphical Toolbox. В нем демонстрируются основные возможности и функции этого набора. Кроме того, приводится пример небольшого приложения на Java, с помощью которого можно просмотреть созданную панель. В этом примере пользователь вводит данные 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

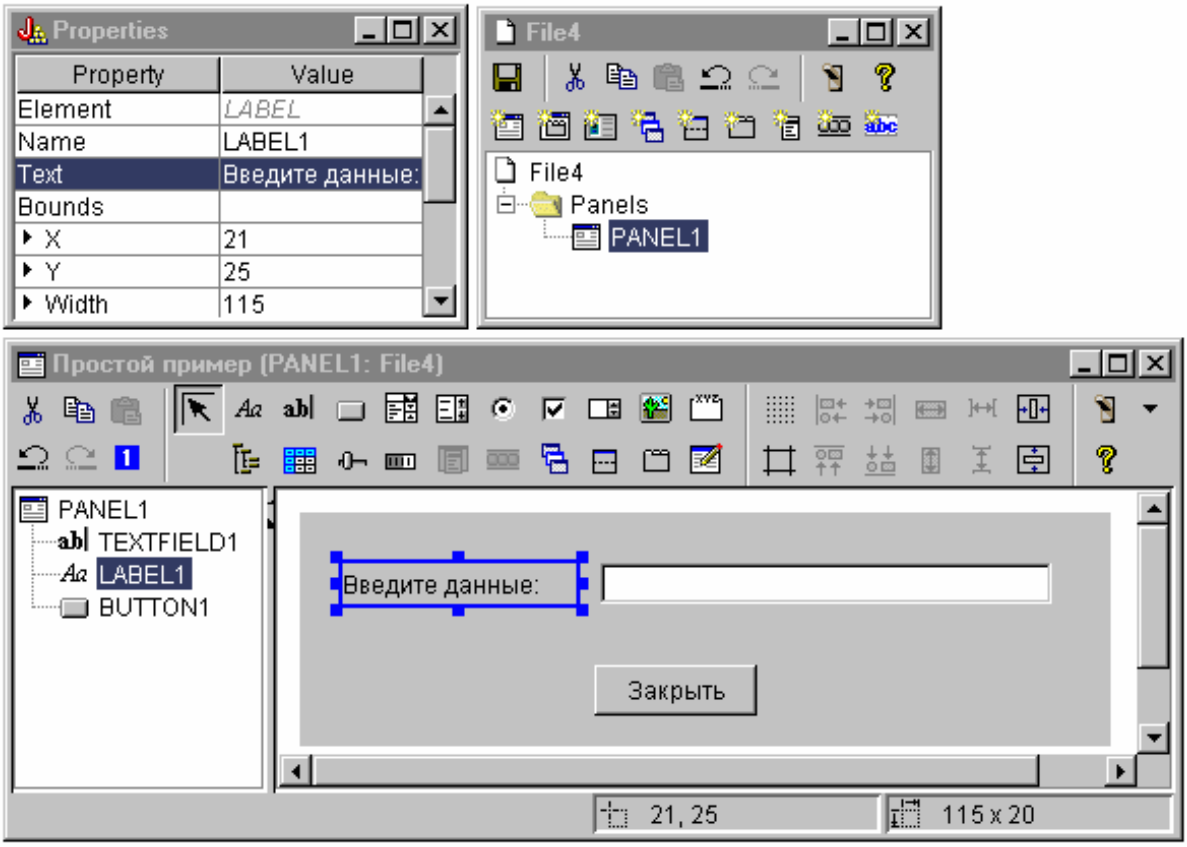
помощью мыши и нажмите клавишу ″Delete″. Добавьте метку, текстовое поле и кнопку с помощью Редактора панелей, расположив их на панели, как показано на рис. 1.

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″.

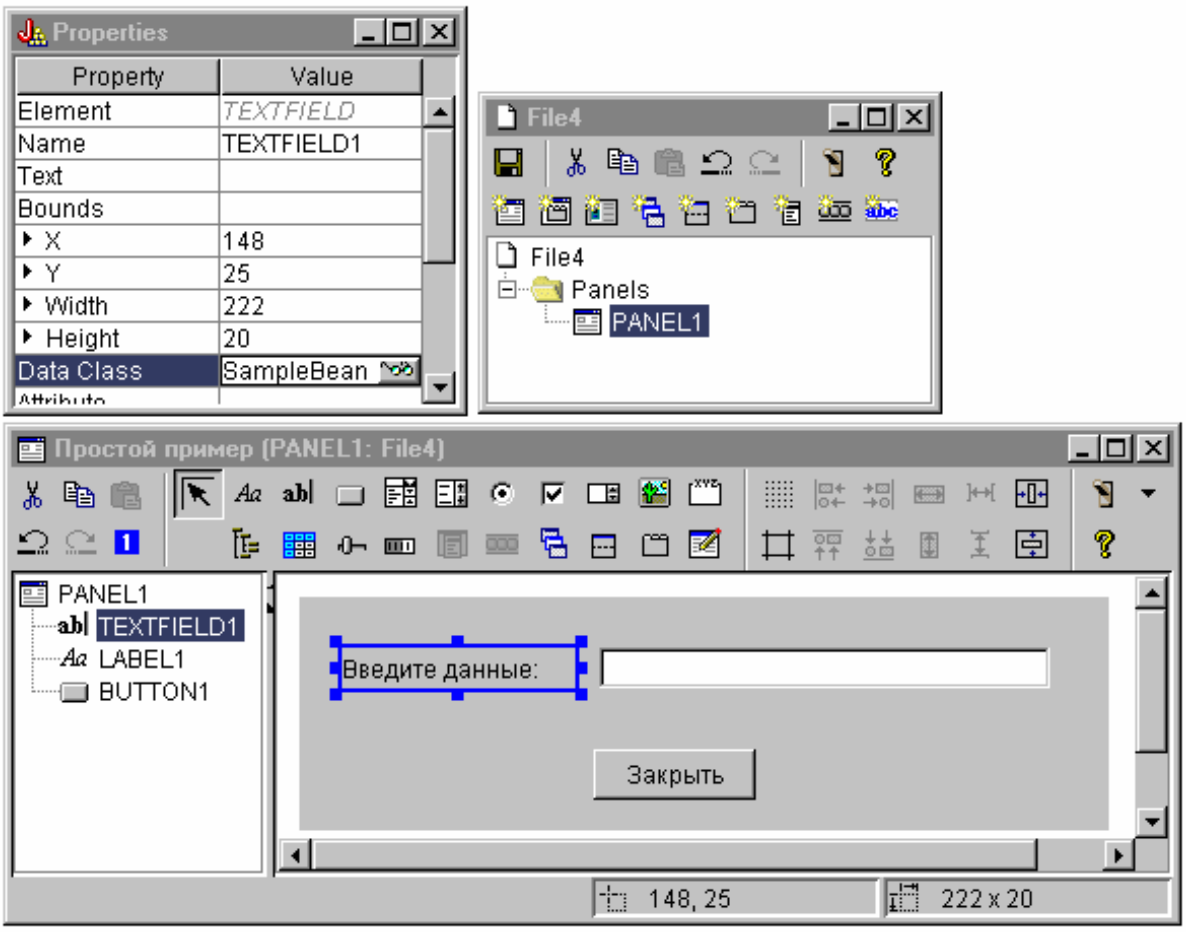
Рисунок 2: Окна GUI Builder: Изменение текста в окне свойств



Текстовое поле

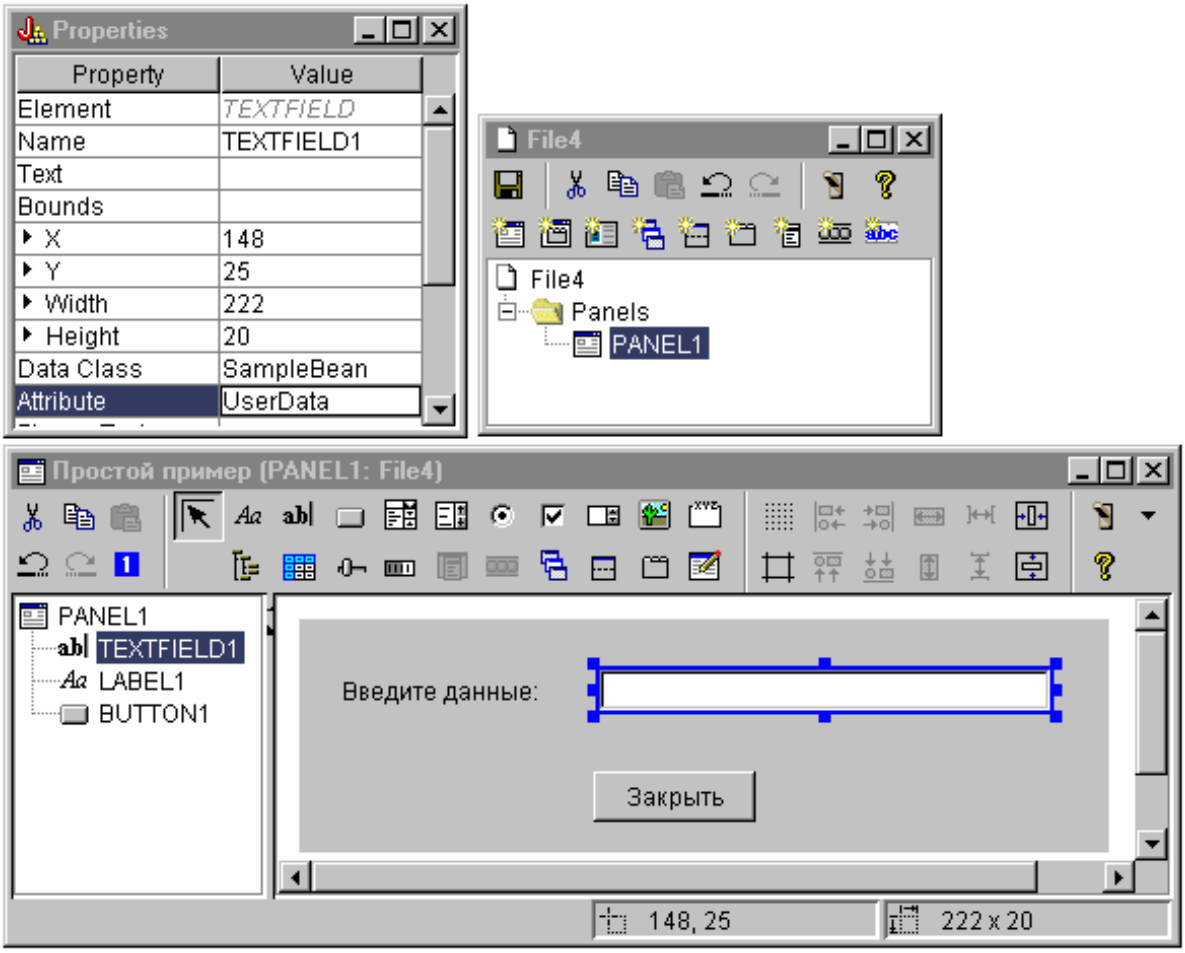
Текстовое поле будет содержать данные. Для того чтобы они были обработаны GUI Builder, измените некоторые свойства этого поля. В поле Класс данных укажите имя класса компонента **SampleBean**. Этот компонент будет содержать данные для текстового поля.

Рисунок 3: Окна GUI Builder: Установка свойства Класс данных



В поле Атрибут укажите свойство компонента, содержащее данные. В данном случае это **UserData**.

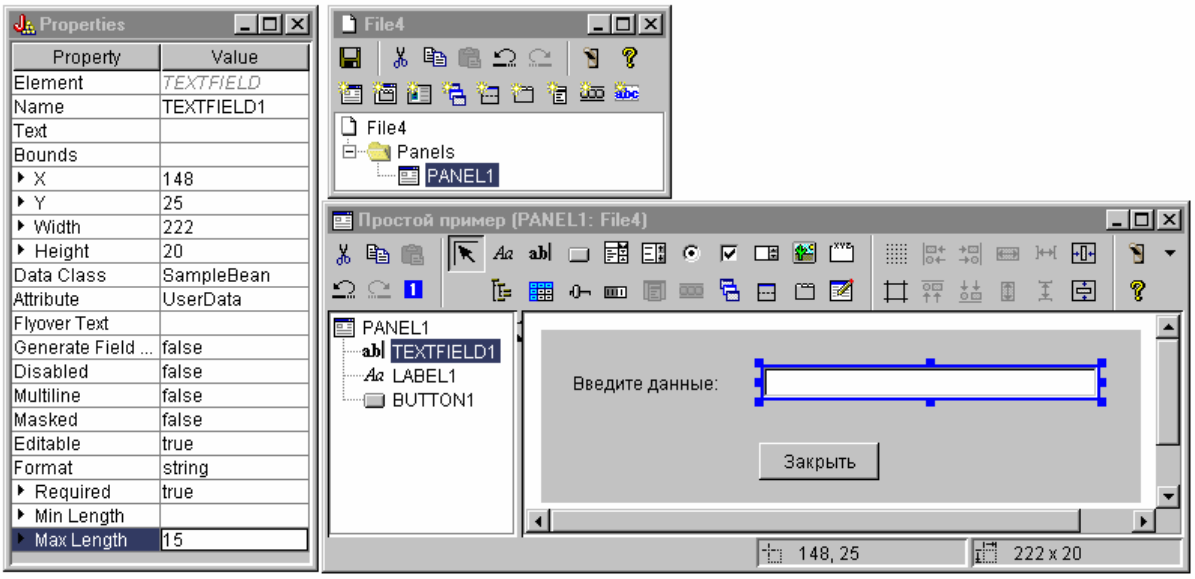
Рисунок 4: Окна GUI Builder: Установка свойства Атрибут



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**.

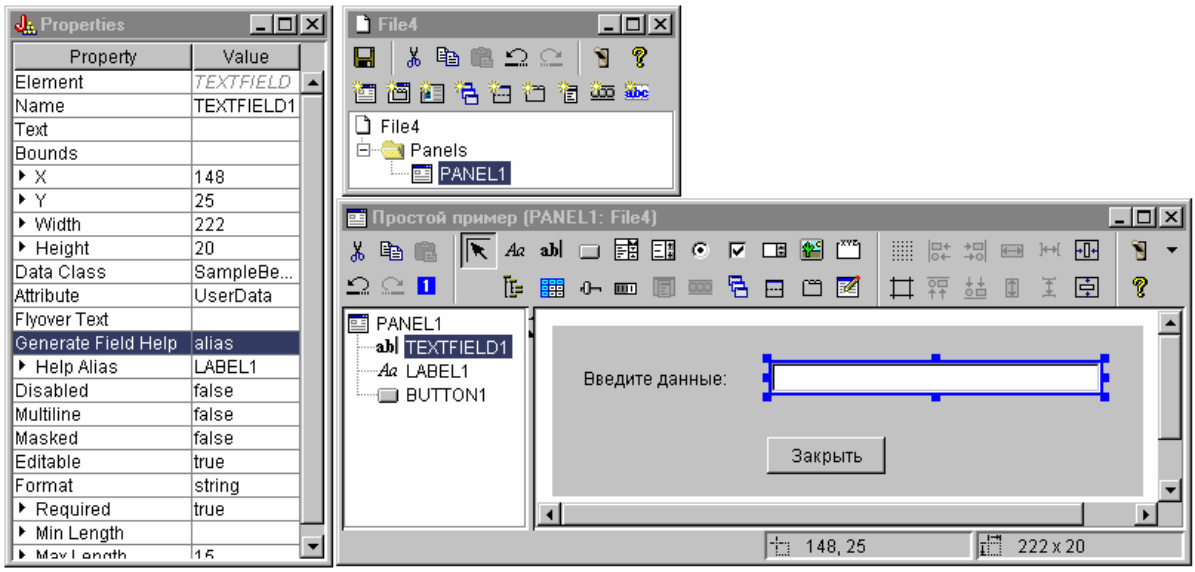
Укажите, что пользователь должен ввести строку, содержащую не больше 15 символов.

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″.

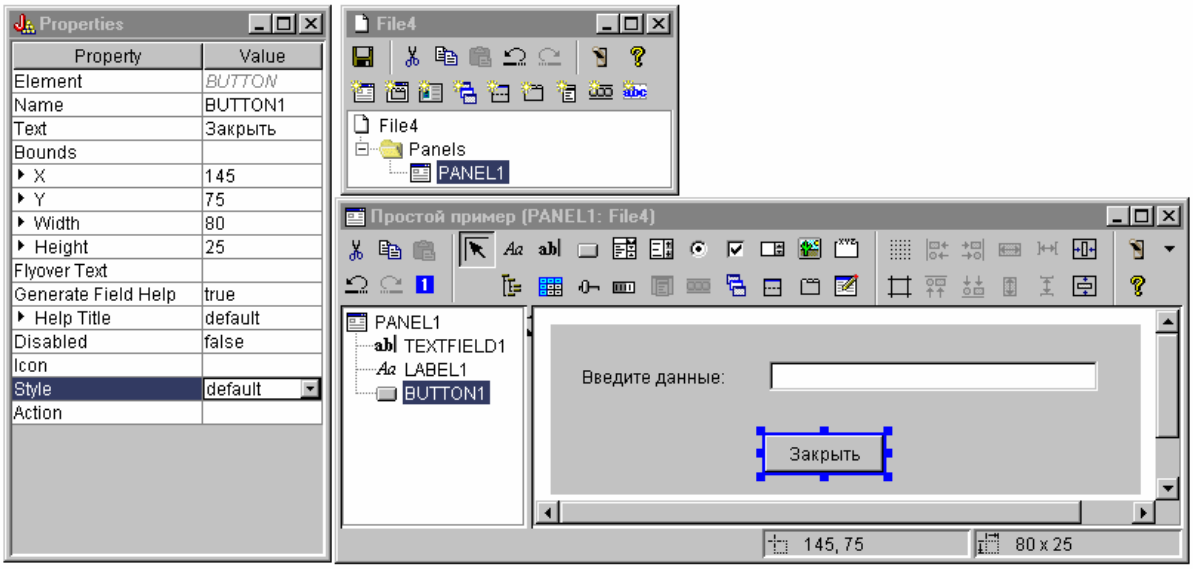
Рисунок 6: Окна GUI Builder: Задание контекстной справки для текстового поля



Кнопка

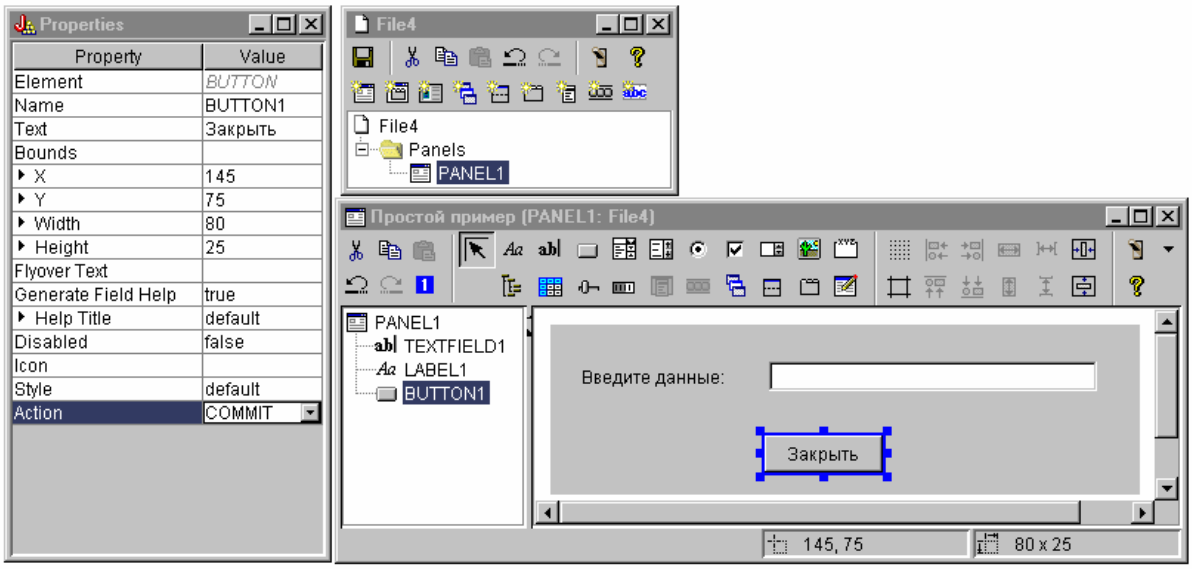
Измените свойство, задающее стиль, выбрав выделение по умолчанию.

Рисунок 7: Окна GUI Builder : Установка выделения кнопок по умолчанию с помощью свойства Стиль



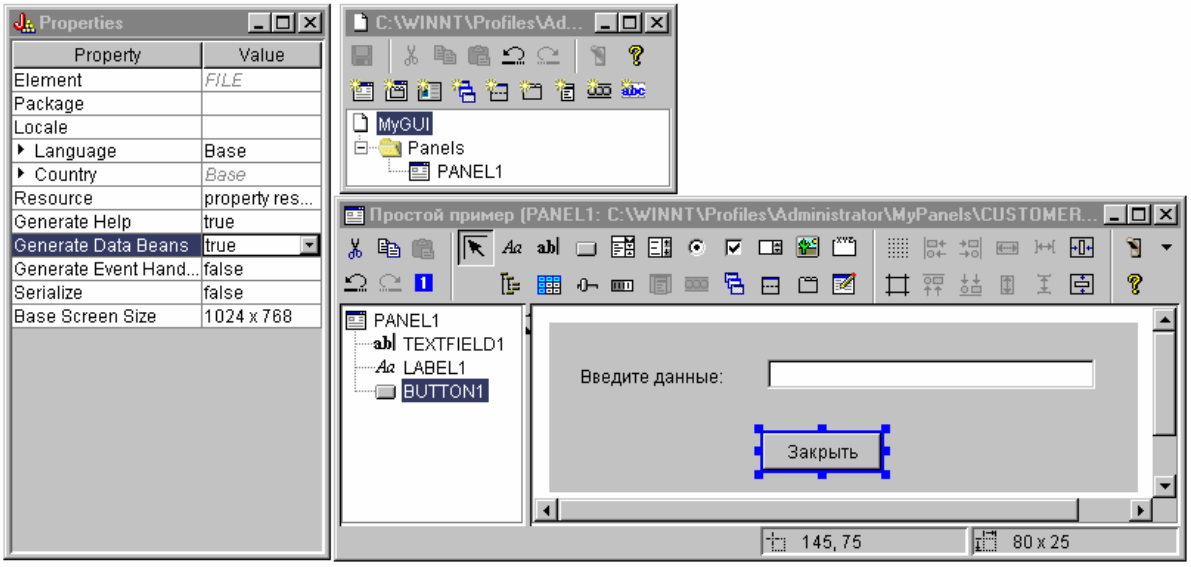
Присвойте свойству Действие значение COMMIT - тогда при нажатии кнопки будет вызываться метод компонента `setUserData`.

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**.

Рисунок 9: Окна GUI Builder: Установка свойств для создания электронного макета справки и объекта JavaBean



Созданные файлы

Сохранив определение панели, просмотрите файлы, созданные GUI Builder. **Файл PDML** Ниже приведено содержимое файла **MyGUI.pdml**, который поможет вам понять основы языка PDML. Поскольку вы работаете с PDML только с помощью инструментов Graphical Toolbox, вы можете не изучать формат файла подробно:

```
<!-- 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 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.

GUI Builder уже вызвал компилятор Java для структуры справки и создал соответствующий файл класса. При работе с данным примером можно не изменять реализацию компонента. В практическом приложении Java, как правило, изменяются методы `загрузки` и `сохранения` данных для внешних источников. В

большинстве случаев применяется стандартная реализация двух других методов. Дополнительная информация приведена в документации по интерфейсу **DataBean**, содержащейся в javadocs для среды выполнения PDML.

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. Вам достаточно создать новый исходный файл Java, который будет содержать главную точку входа для приложения. В этом примере файл называется **SampleApplication.java**. Он содержит следующий код:

```
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);
    }
}
```

Как правило, инициализацию компонента или компонентов выполняет вызывающая программа с помощью методов **загрузки**. Если данные для панели содержатся в нескольких компонентах, то каждый из них необходимо инициализировать перед передачей в среду Graphical Toolbox.

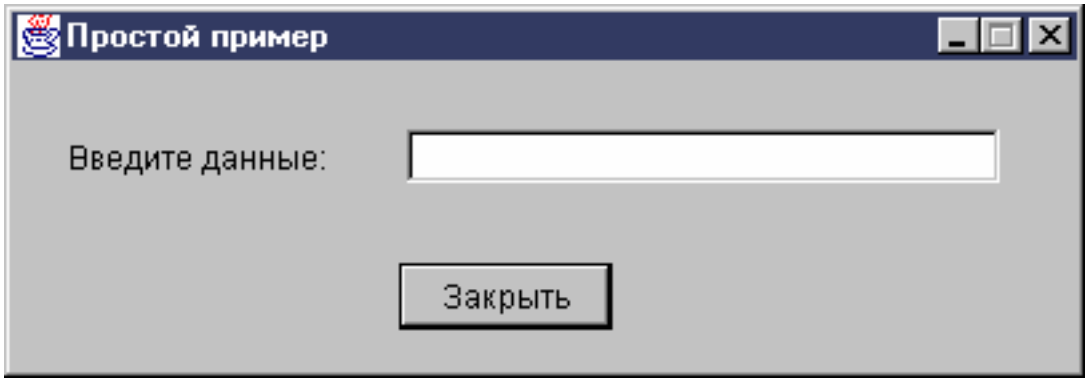
Класс **com.ibm.as400.ui.framework.java.PanelManager** содержит API, позволяющий просматривать отдельные окна и окна диалогов. Graphical Toolbox обрабатывает имя файла PDML в конструкторе как имя ресурса - каталог, а также файл ZIP или JAR, содержащие файл PDML, необходимо указать в переменной CLASSPATH.

Поскольку объект **Фрейм** входит в состав конструктора, окно будет обладать всеми свойствами модального окна диалога. В практической программе на Java этот объект может быть получен с помощью подходящего родительского окна диалога. Поскольку окно является модальным, управляющий элемент не будет возвращен в приложение до закрытия окна. После закрытия приложение передает измененные данные пользователя и завершает работу.

Запуск приложения

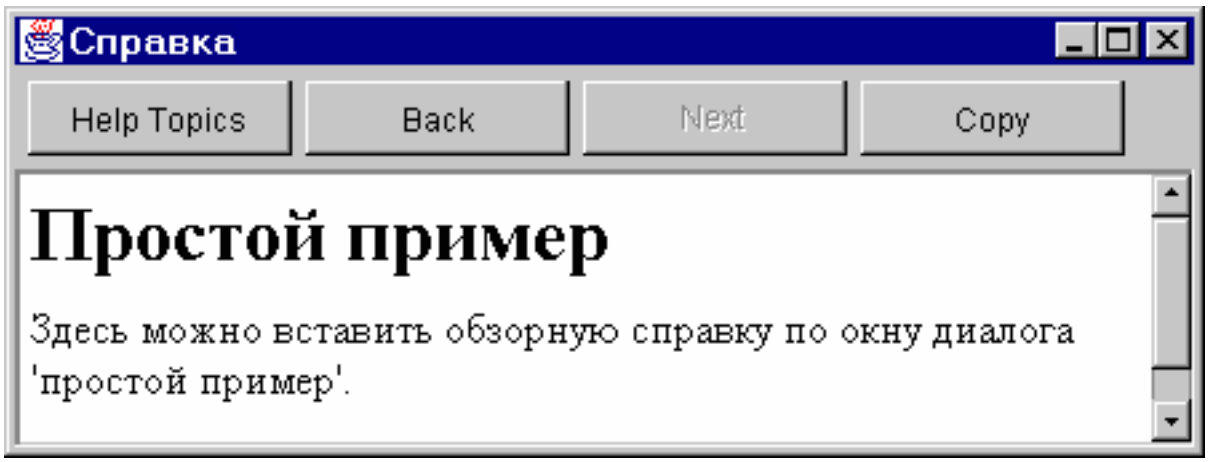
Ниже приведен примерный вид окна, которое будет открыто при запуске приложения:

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.

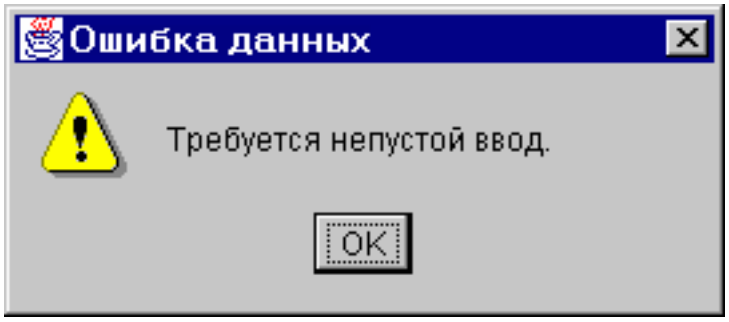
Рисунок 11: Структура электронной справки простого примера



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.

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

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);
        }
    }


}
```

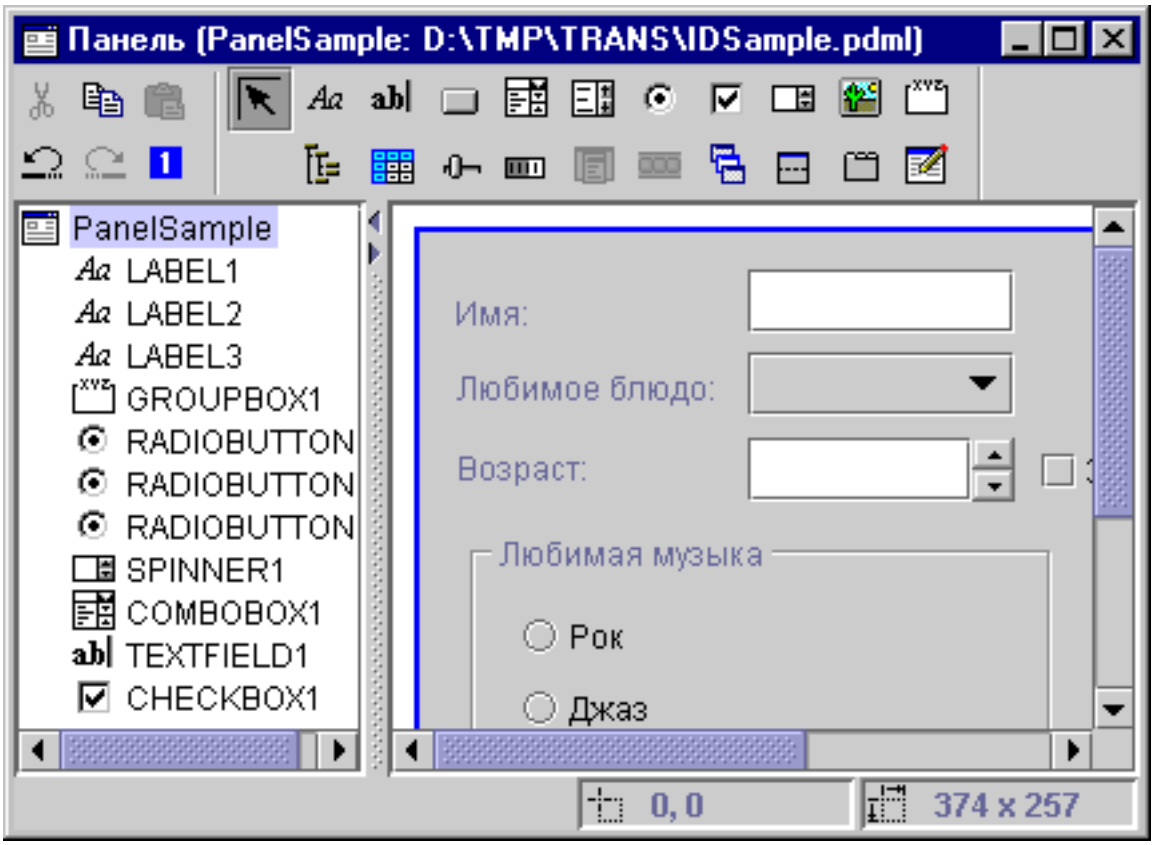
Создание панели с помощью GUI Builder

GUI Builder значительно упрощает процедуру создания панели. В строке меню главного окна GUI Builder выберите **Файл → Создать файл**.

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.

Рисунок 1: Создание простой панели с помощью 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";
    }
}
```

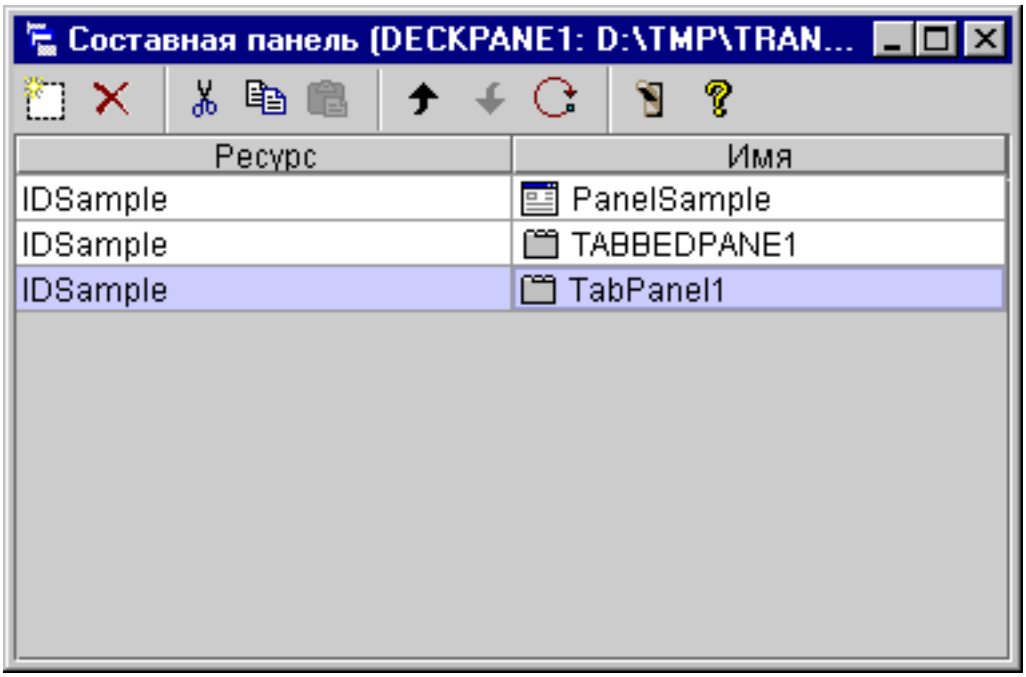
Панель - это один из самых простых компонентов интерфейса, которые можно создать с помощью GUI Builder. Однако даже на основе панели можно создать сложное приложение с удобным пользовательским интерфейсом.

Создание составной панели с помощью GUI Builder

GUI Builder позволяет быстро создать составную панель. В строке меню окна GUI Builder выберите **Файл** -- **Создать файл**.

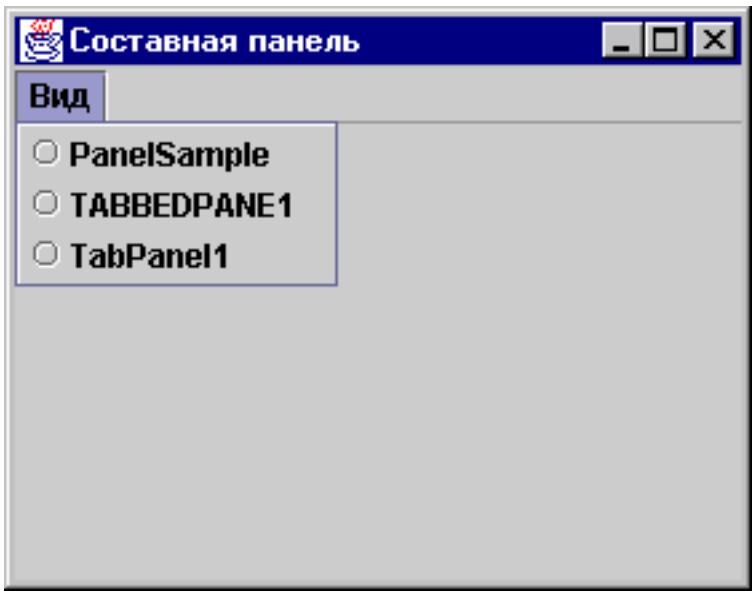
В строке меню окна **Файл** GUI Builder нажмите кнопку **Вставить разделенную панель**  , чтобы перейти к окну создания панелей, в котором можно добавить нужные компоненты составной панели. В приведенном ниже примере добавляется три компонента.

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

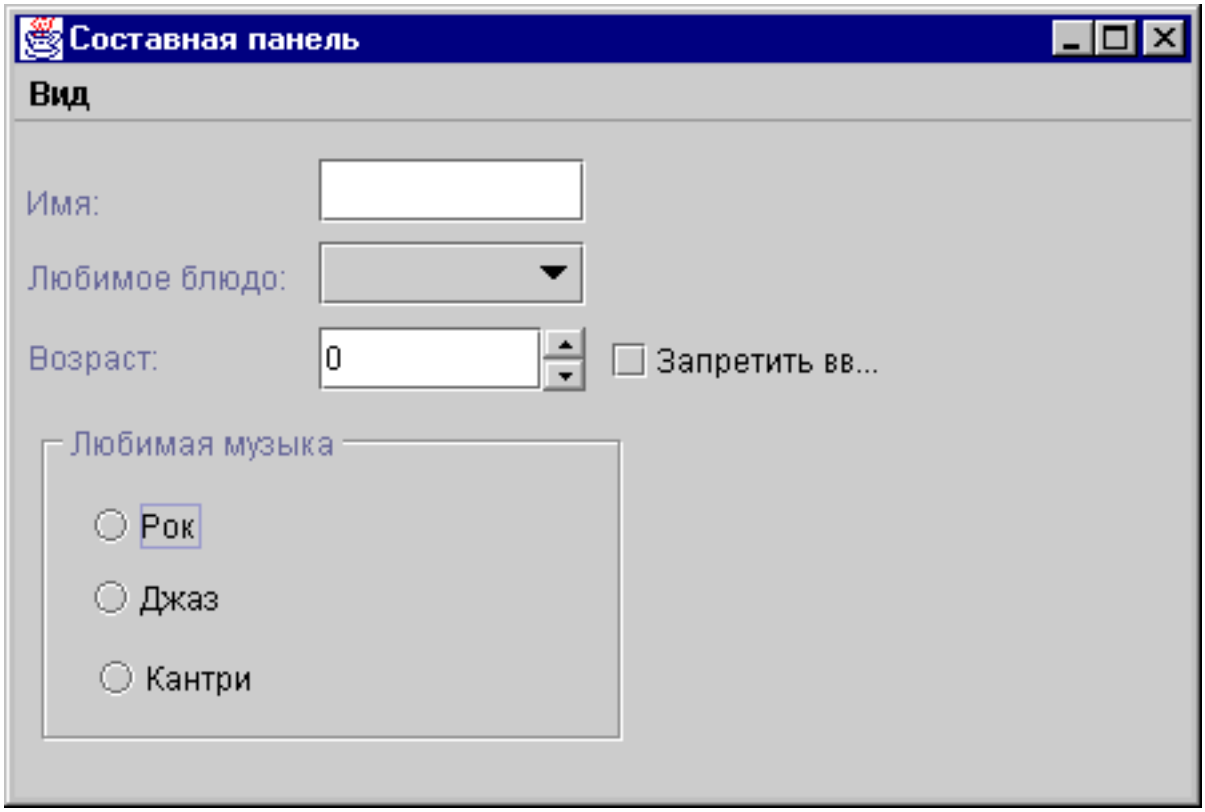


Рисунок 4: Просмотр TABBEDPANE1 с помощью GUI Builder

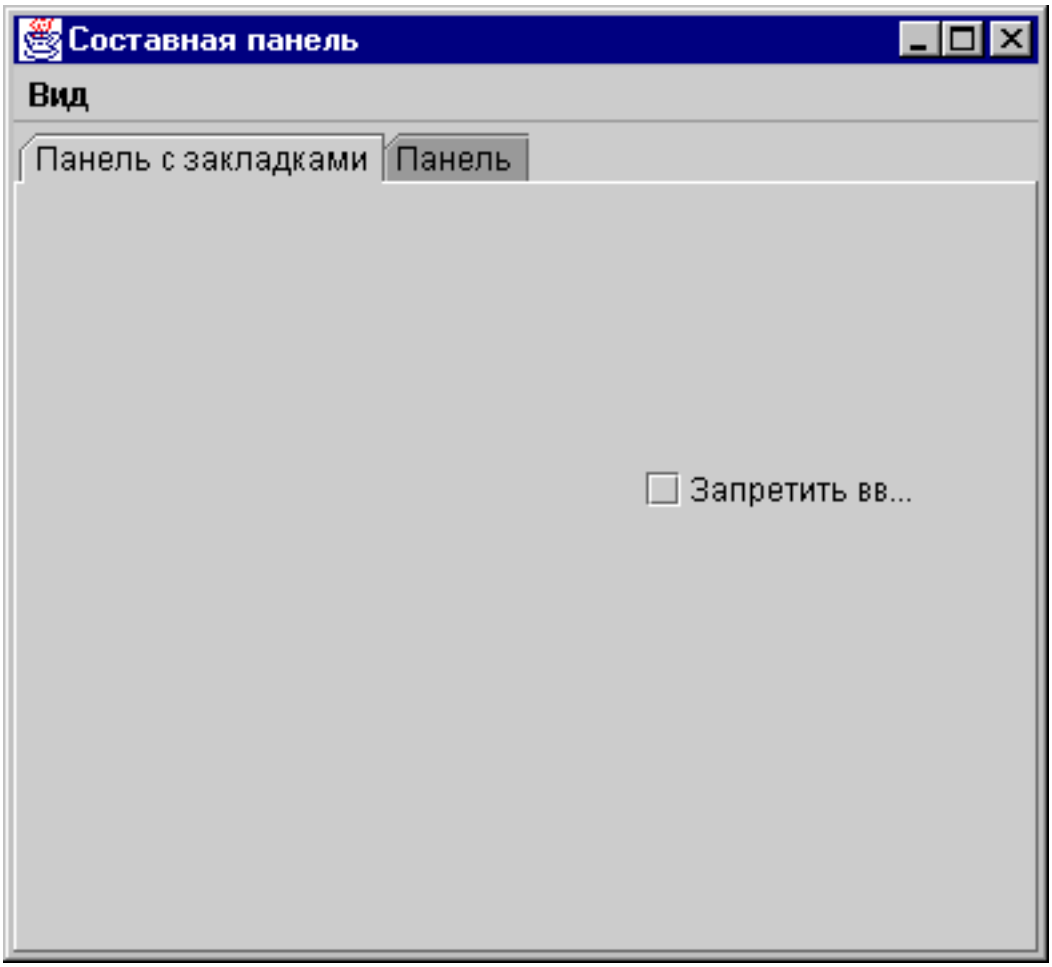
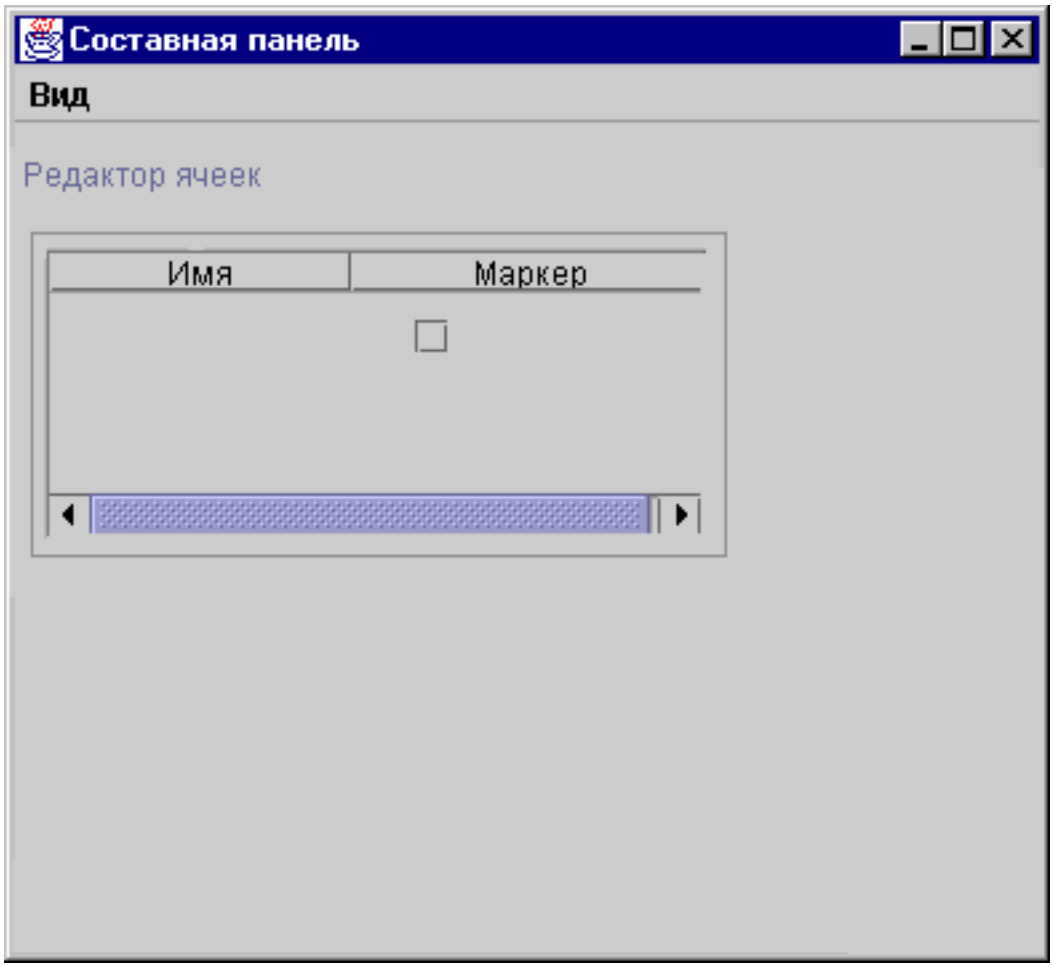


Рисунок 5: Просмотр элемента TablePanel с помощью GUI Builder

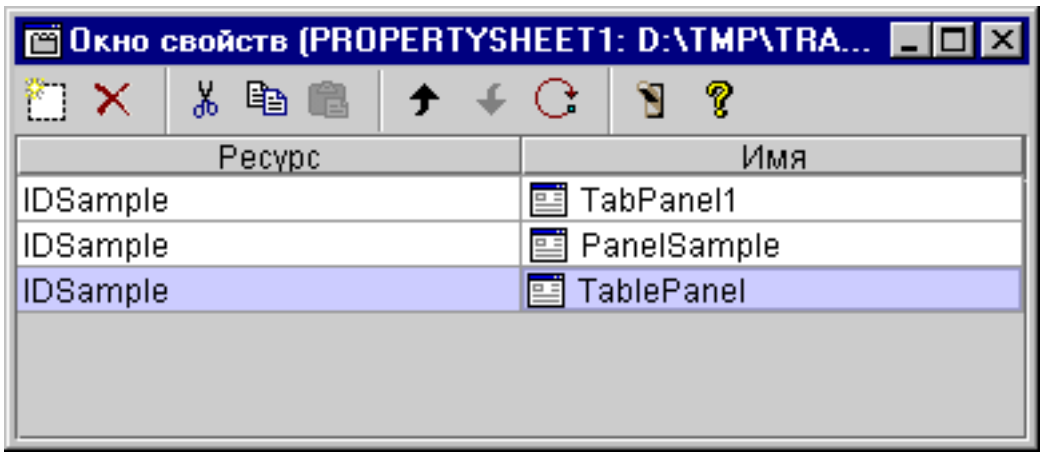


Создание окна свойств с помощью GUI Builder

Программа GUI Builder позволяет упростить процедуру создания окна свойств. В строке меню GUI Builder выберите **Файл --> Создать файл**.

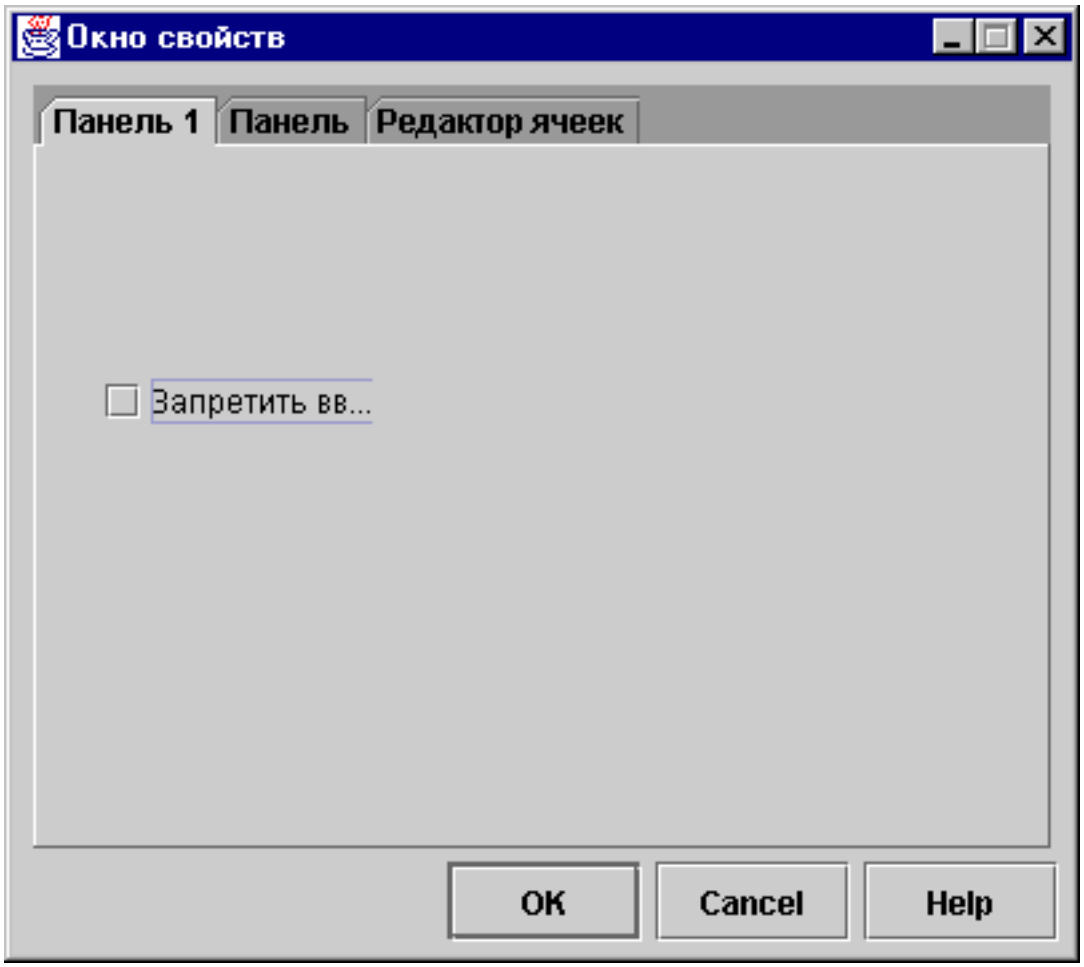
В строке меню окна **Файл** GUI Builder нажмите кнопку Вставить разделенную панель  . Появится Редактор панелей, в котором можно можно добавить компоненты разделенной панели.

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

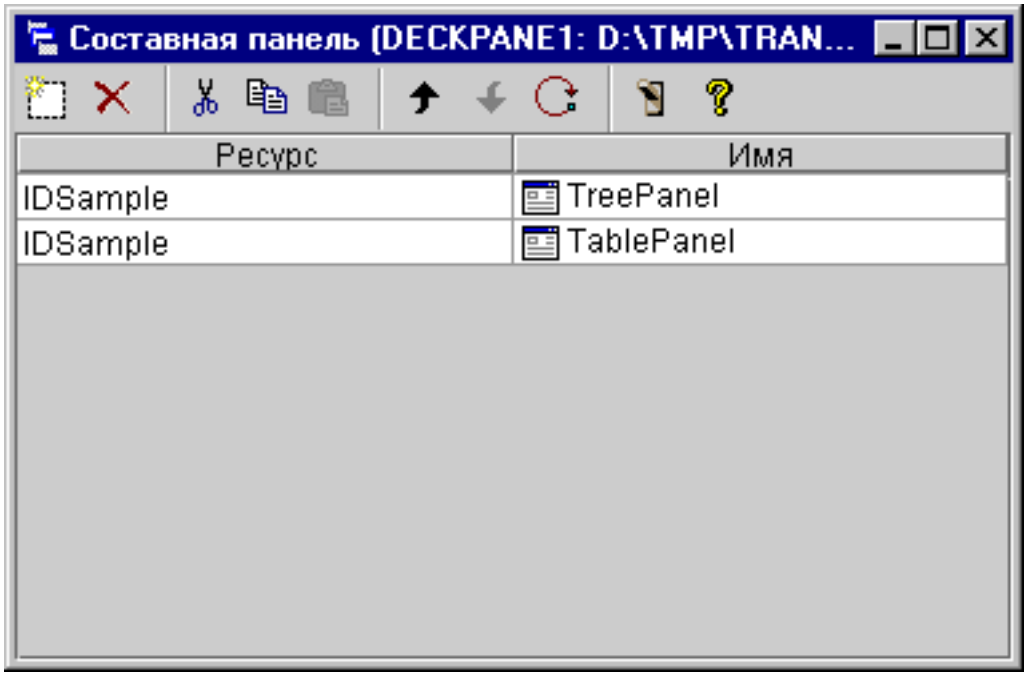


Создание разделенной панели в GUI Builder

Программа GUI Builder упрощает создание разделенных панелей. В строке меню GUI Builder выберите **Файл --> Создать файл**.

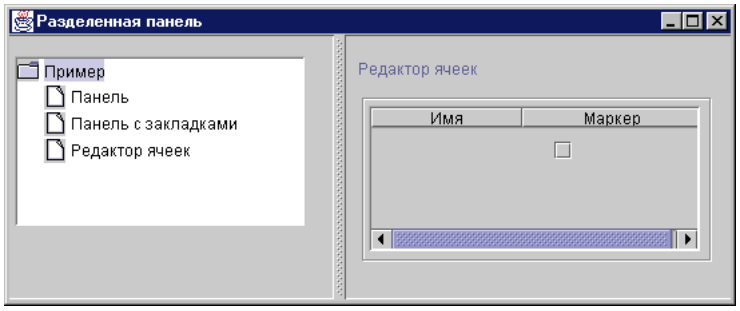
В меню **Файл** GUI Builder щелкните на значке Вставить новую панель  для перехода к окну создания панелей, в котором можно добавлять новые компоненты разделенной панели. В следующем примере добавляются два компонента.

Рисунок 1: Создание разделенной панели с помощью GUI Builder



После создания разделенной панели нажмите кнопку **Предварительный просмотр**  для ее просмотра (см. рисунок 2).

Рисунок 2: Предварительный просмотр разделенной панели с помощью GUI Builder

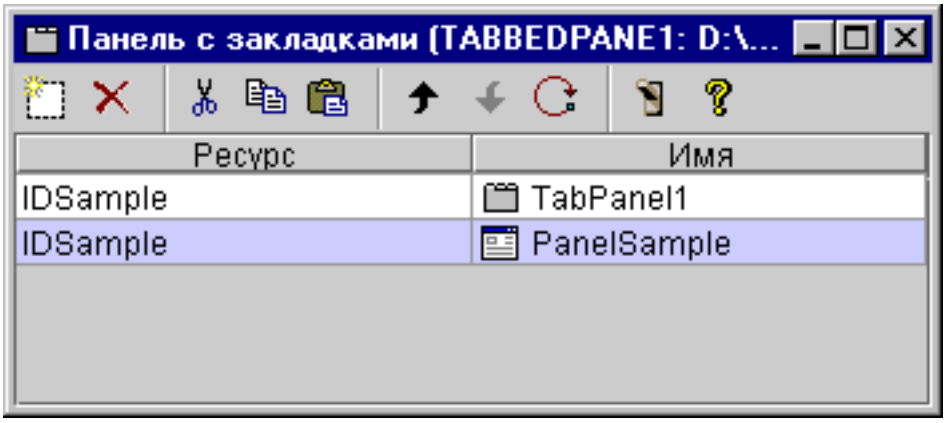


Создание панели со вкладками с помощью GUI Builder

Программа GUI Builder упрощает создание панелей со вкладками. В строке меню GUI Builder выберите **Файл --> Создать файл**.

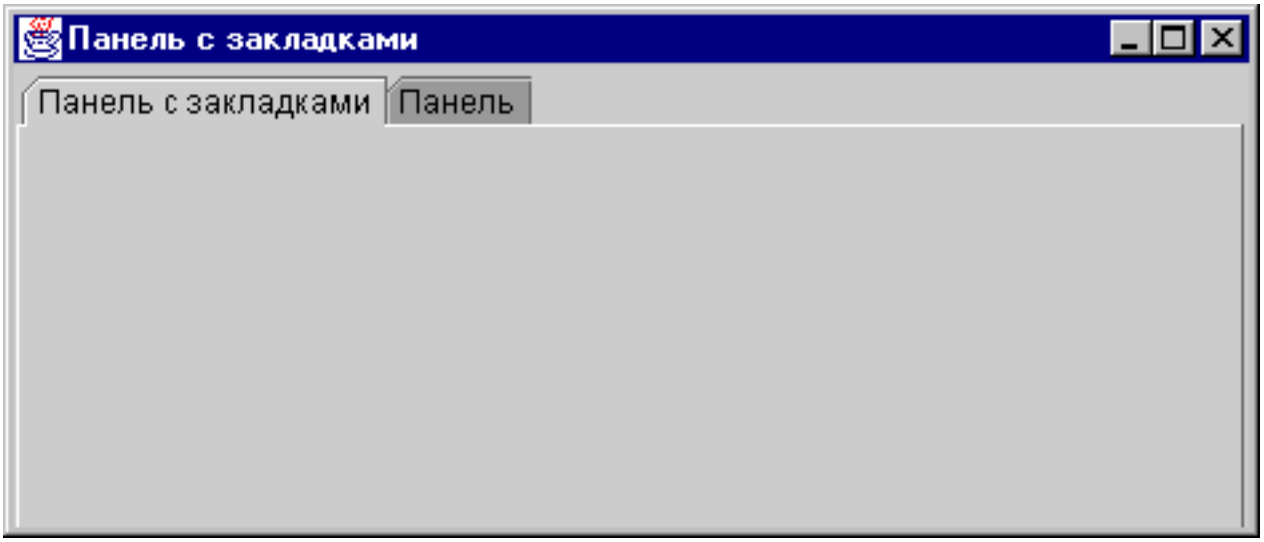
В меню **Файл** GUI Builder щелкните на значке Вставить панель с закладками  для перехода к окну создания панели, в котором в панель можно добавить новые компоненты. В следующем примере добавляются два компонента.

Рисунок 1: Создание панели с закладками с помощью GUI Builder



После создания панели с закладками нажмите кнопку **Предварительный просмотр** для просмотра панели.

Рисунок 2: Предварительный просмотр панели с закладками с помощью GUI Builder



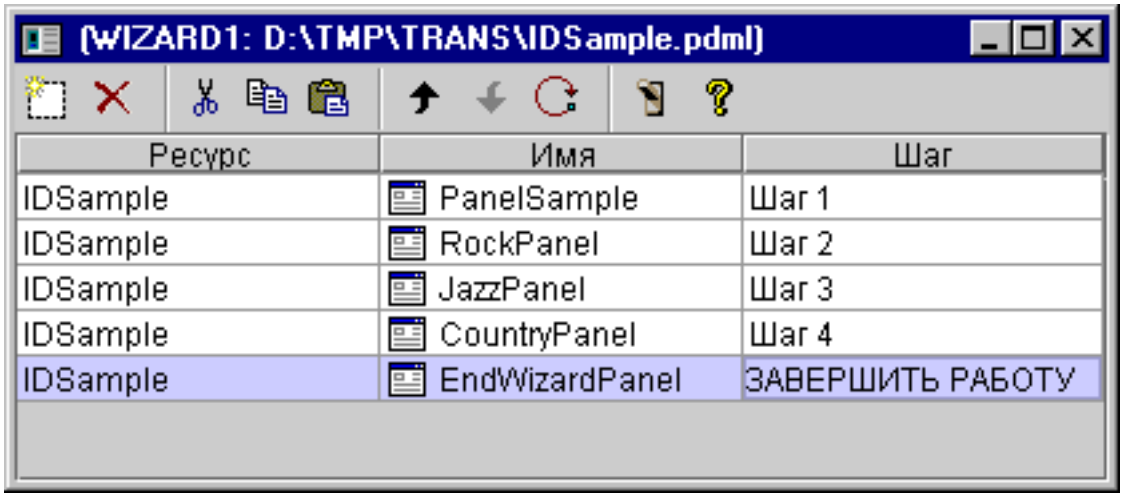
Создание мастера с помощью GUI Builder

Программа GUI Builder упрощает создание программ-мастеров. В строке меню GUI Builder выберите **Файл --> Создать файл**.

В строке меню окна **Файл** программы GUI Builder нажмите на кнопку Вставить мастер

. Появится окно создания мастера.

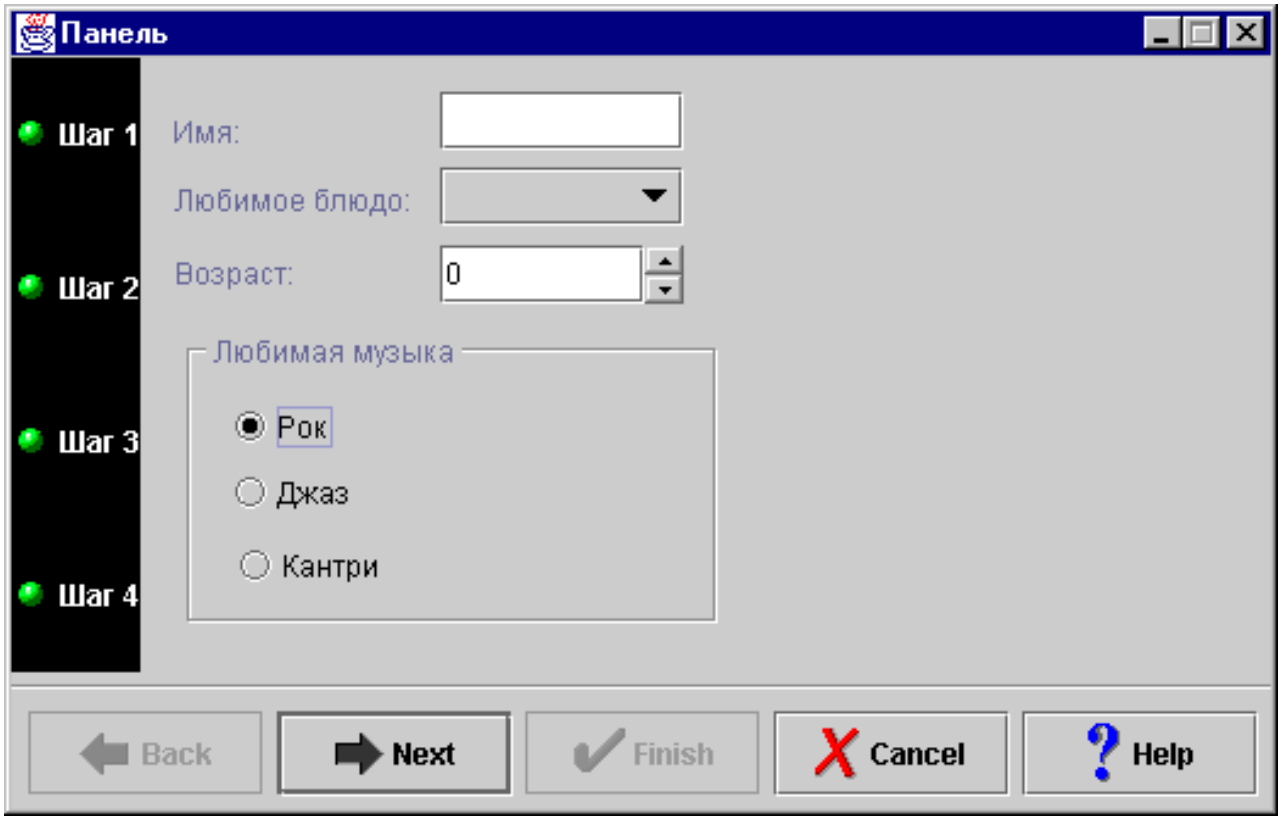
Рисунок 1: Создание мастера с помощью GUI Builder



После создания мастера нажмите кнопку **Предварительный просмотр**

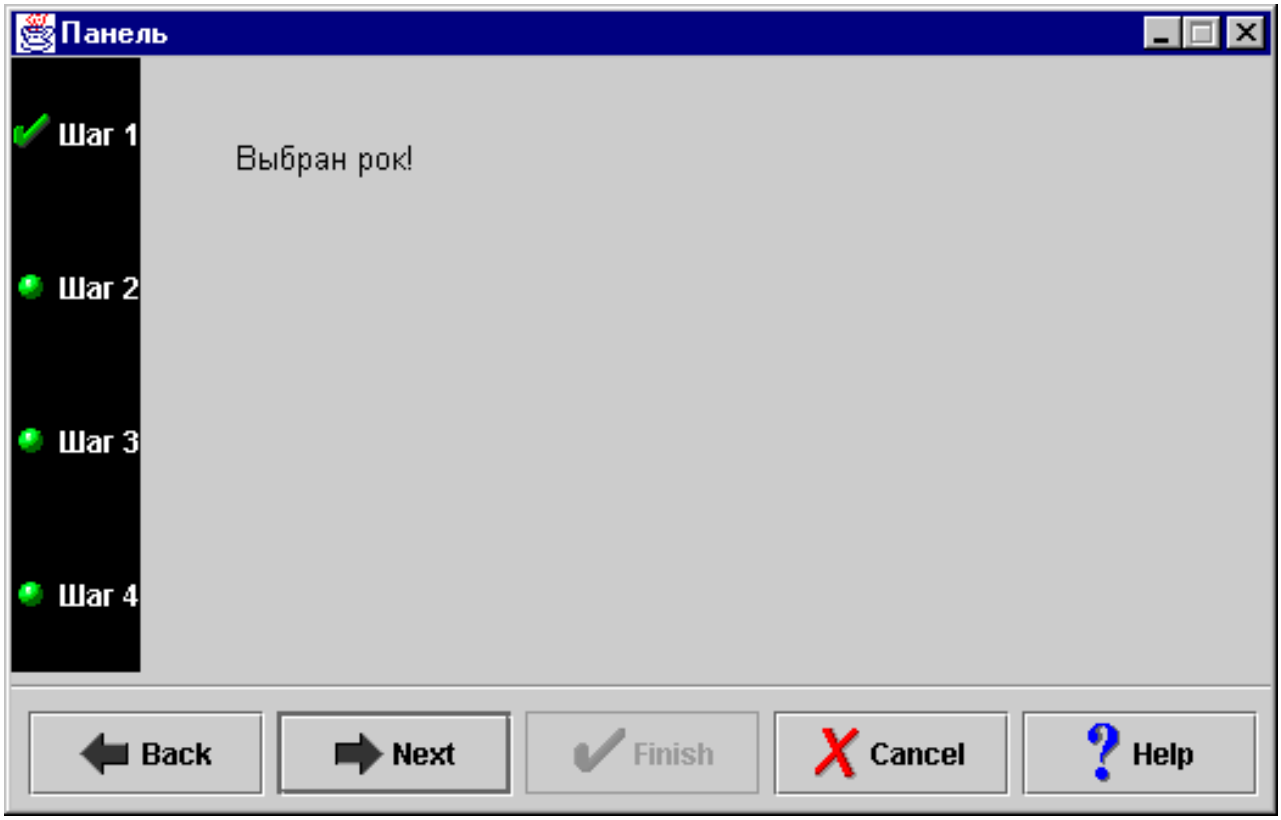
для его просмотра. На рис. 2 показано окно предварительного просмотра для данного примера.

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.

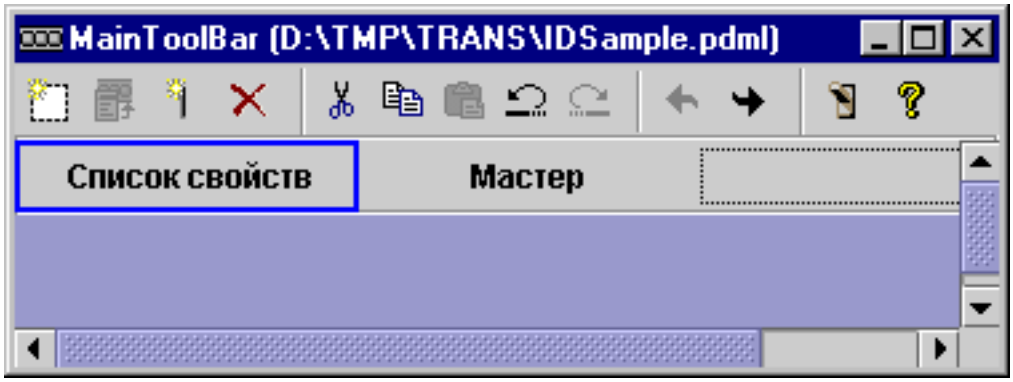
Рисунок 4: Предварительный просмотр последней панели мастера с помощью GUI Builder

Создание панели инструментов с помощью GUI Builder

Программа GUI Builder упрощает создание панелей инструментов. В строке меню GUI Builder выберите **Файл --> Создать файл**.

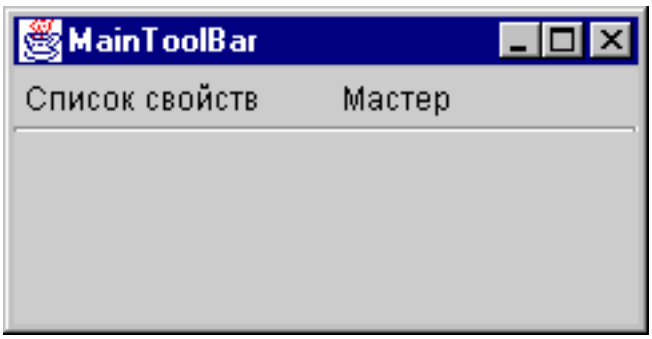
В строке меню окна **Файл** GUI Builder нажмите кнопку **Вставить панель инструментов**. Появится окно создания панели инструментов.

Рисунок 1: Создание панели инструментов с помощью GUI Builder



После создания панели инструментов нажмите кнопку **Предварительный просмотр**  для просмотра панели. В данном случае панель инструментов может вызывать окно свойств или окно мастера.

Рисунок 2: Предварительный просмотр панели инструментов с помощью GUI Builder

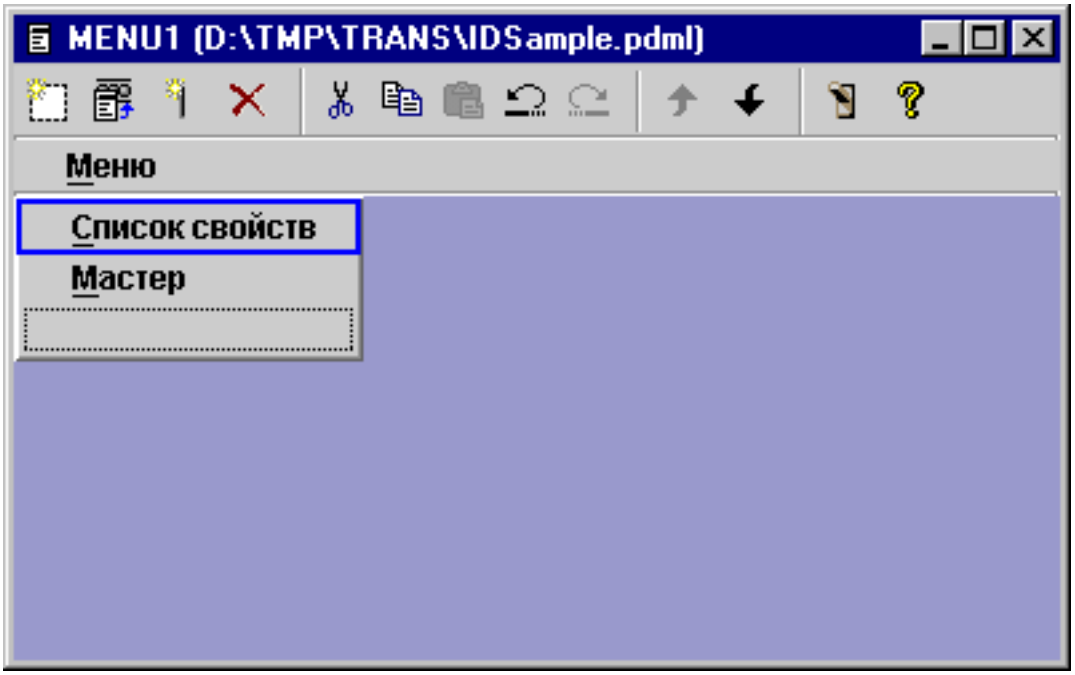


Создание меню с помощью GUI Builder

Программа GUI Builder позволяет легко создавать строки меню. В строке меню GUI Builder выберите **Файл --> Создать файл**.

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

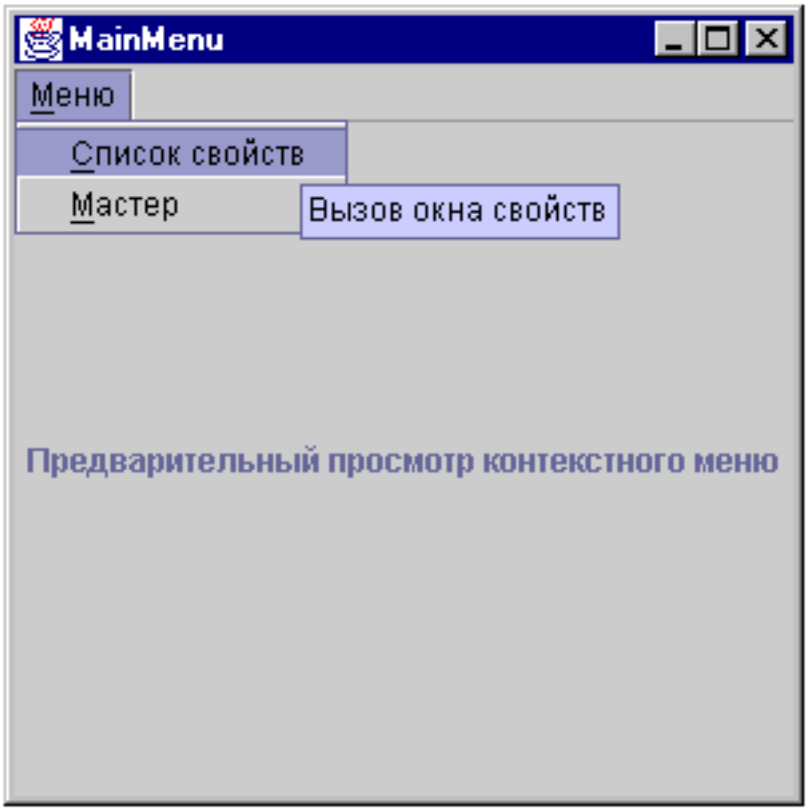
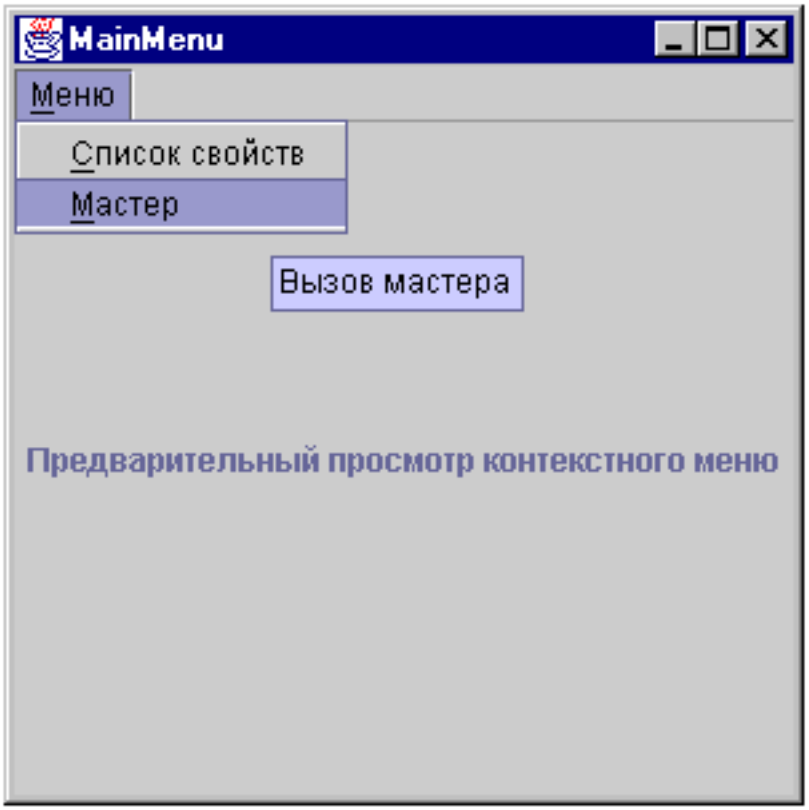


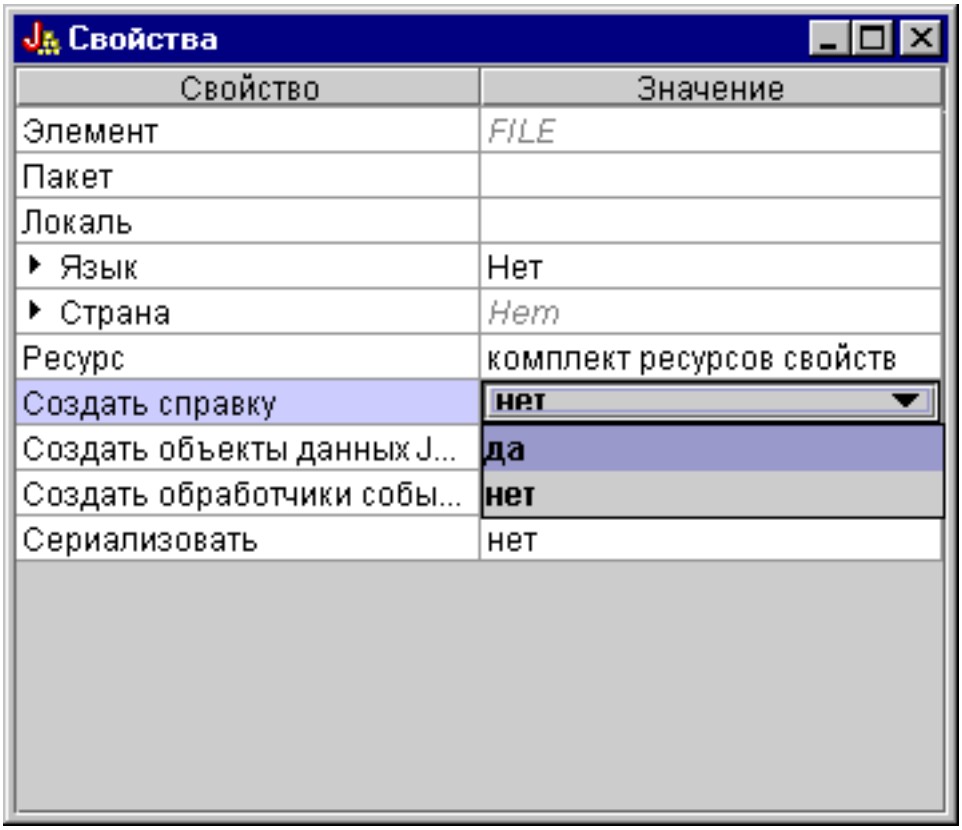
Рисунок 3: GUI Builder - Просмотр Мастера в меню Запуск



Пример: Создание справки

GUI Builder позволяет легко создавать файлы справки. На панели свойств файла, с которым вы работаете, включите опцию ″Создать справку″.

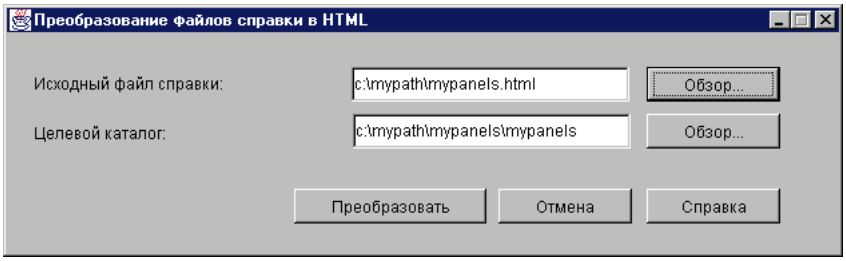
Рисунок 1: Настройка свойства Создать справку на панели Параметры GUI Builder



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:

Рисунок 2: Окно Преобразование документа справки в HTML



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″.

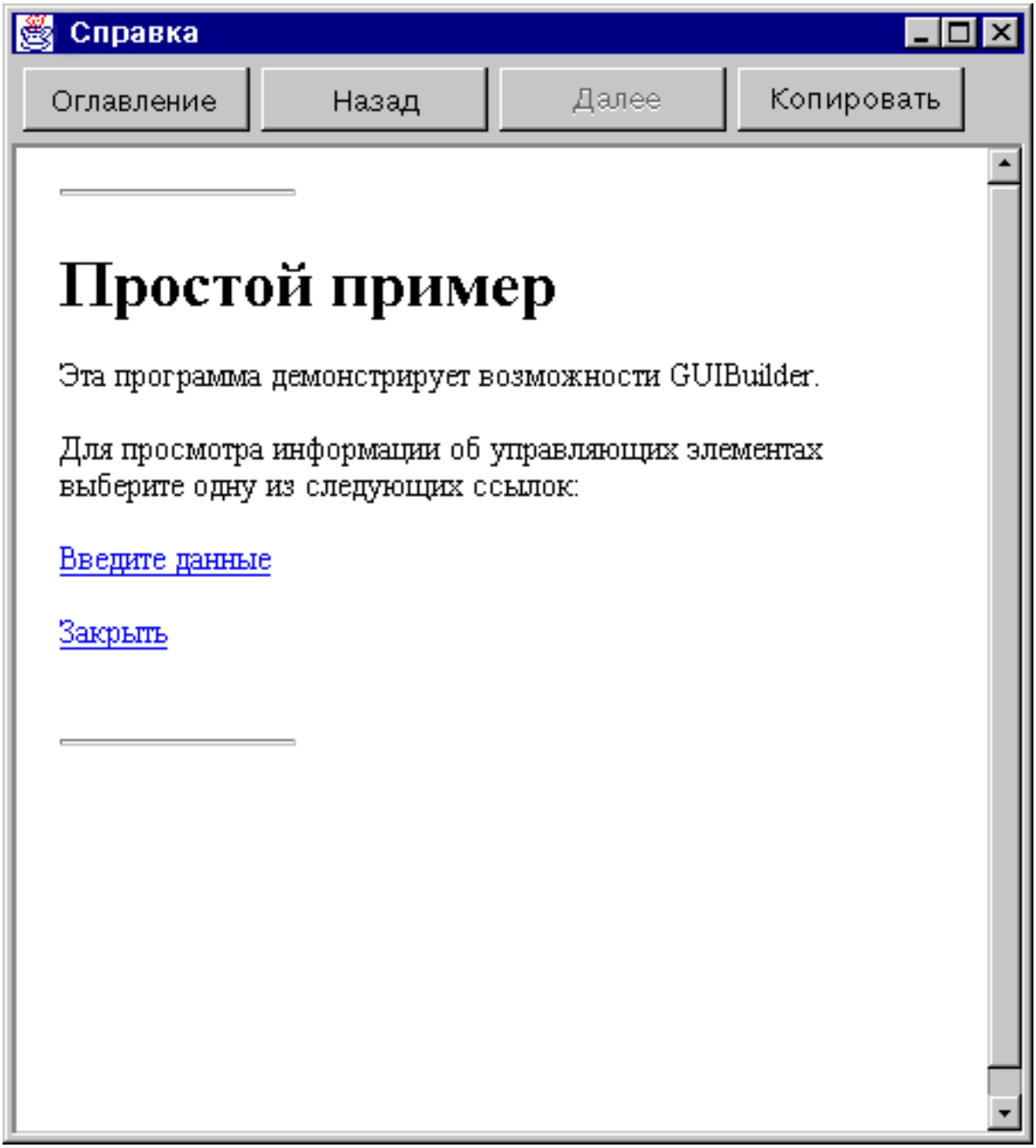
Вы можете разбить исходный файл справки на подразделы, введя в командной строке следующую команду:

```
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

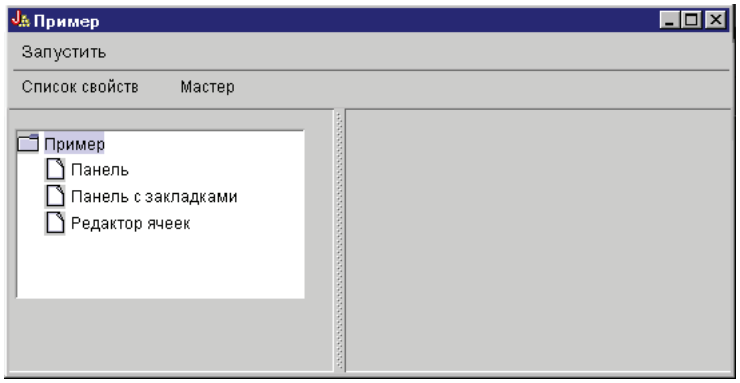


Пример: Применение GUI Builder

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.

Рисунок 1: главное окно GUI Builder



Не забывайте, что вы можете воспользоваться динамическим администратором панели. На рис. 2 и 3 показано, как можно изменить размер окна.

Рисунок 2: Увеличение размера окна GUI Builder

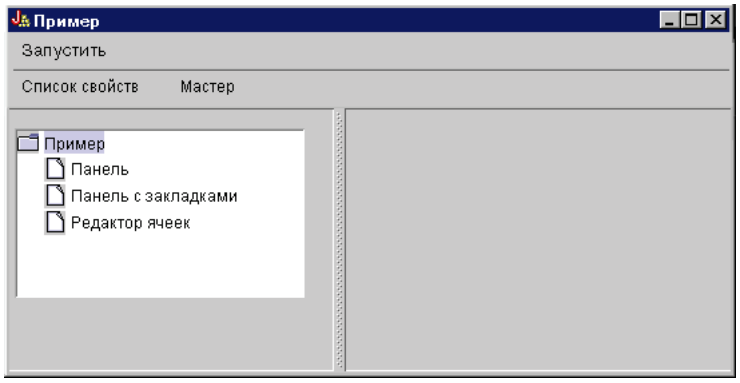
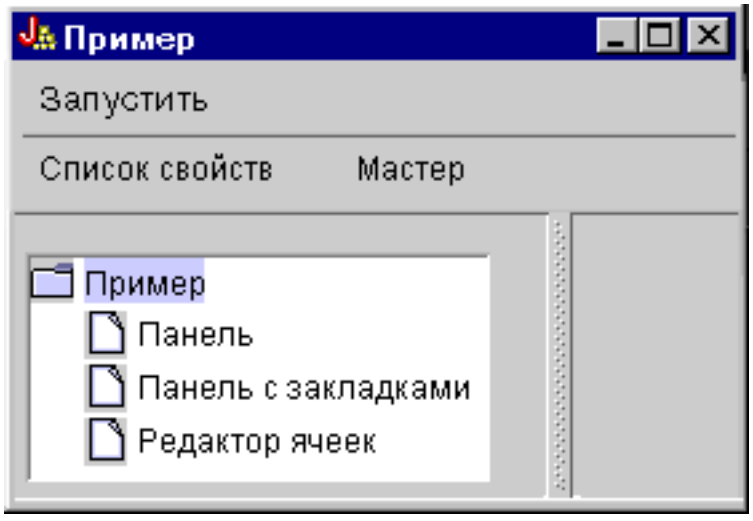


Рисунок 3: уменьшение размера окна GUI Builder



При изменении размера панели и управляющих элементов с помощью динамического администратора панели размер текста не меняется.

На этой панели можно выполнить следующие действия:

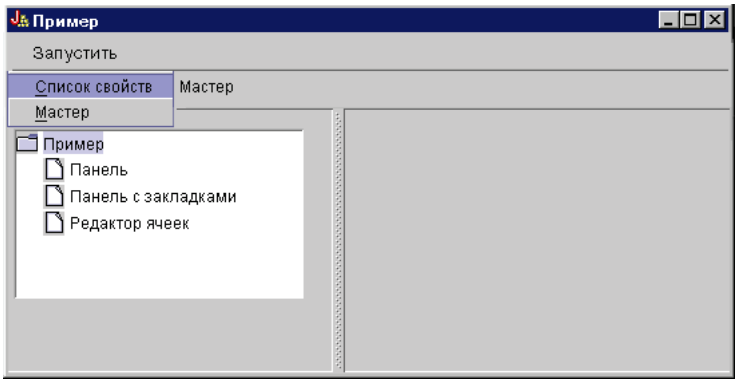
- Открыть окно свойств

- Запустить мастер
- Показать примеры, перечисленные в левой панели

Открыть окно свойств

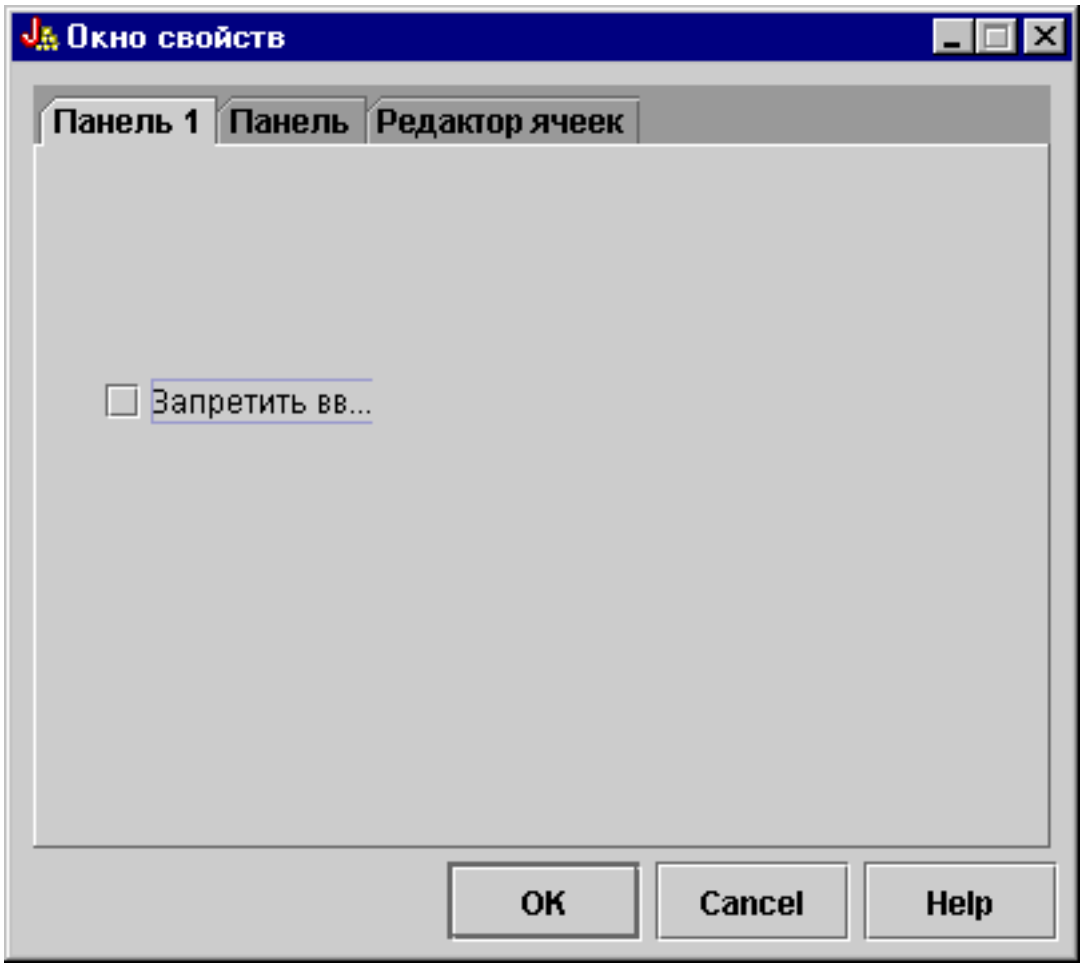
Для создания окна свойств нужно нажать кнопку Окно свойств на панели инструментов или воспользоваться меню **Запустить**. В этом примере демонстрируется связь между пунктами меню и элементами панели инструментов. На рис. 4 показано **окно свойств**, выбранное из меню **Запустить** главного окна GUI Builder.

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. На рис. 6 и 7 показан вид окна после перехода к другим вкладкам.

Рисунок 6: вкладка Пример панели

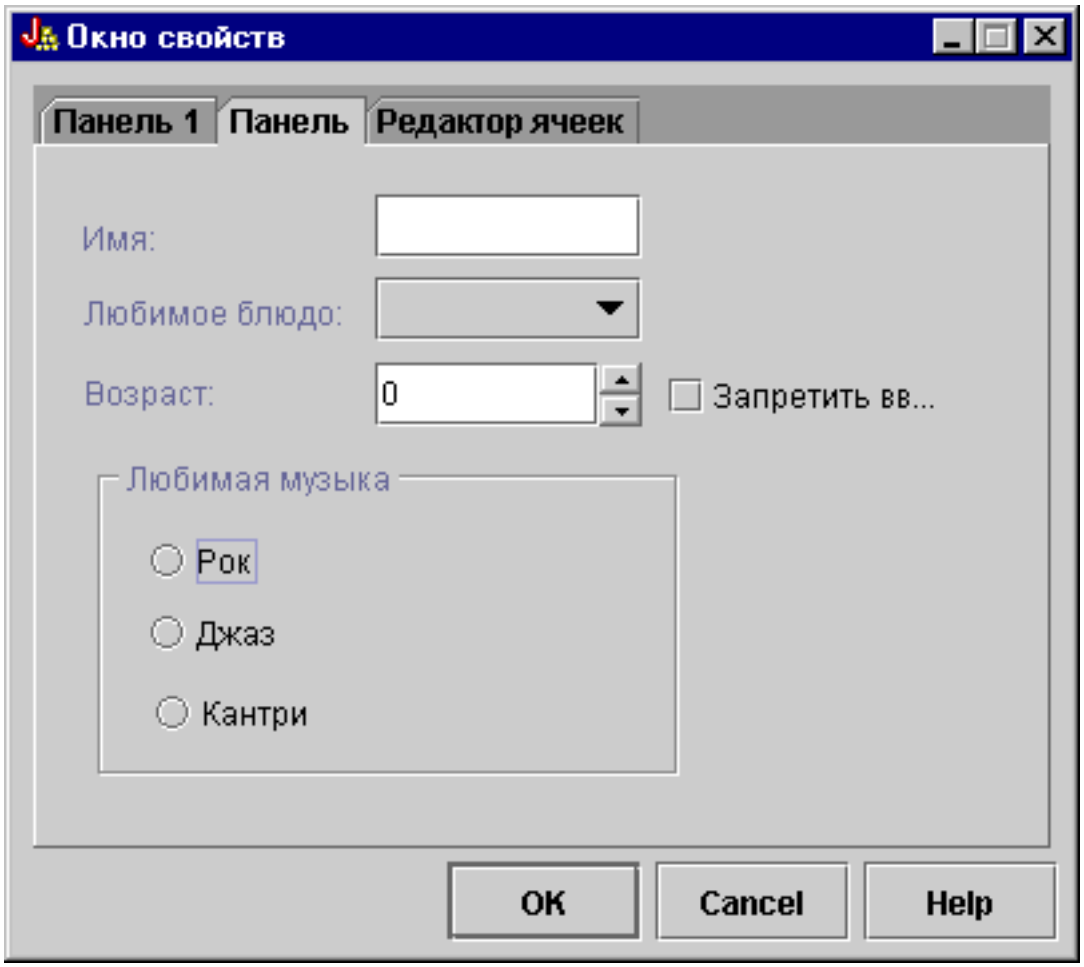
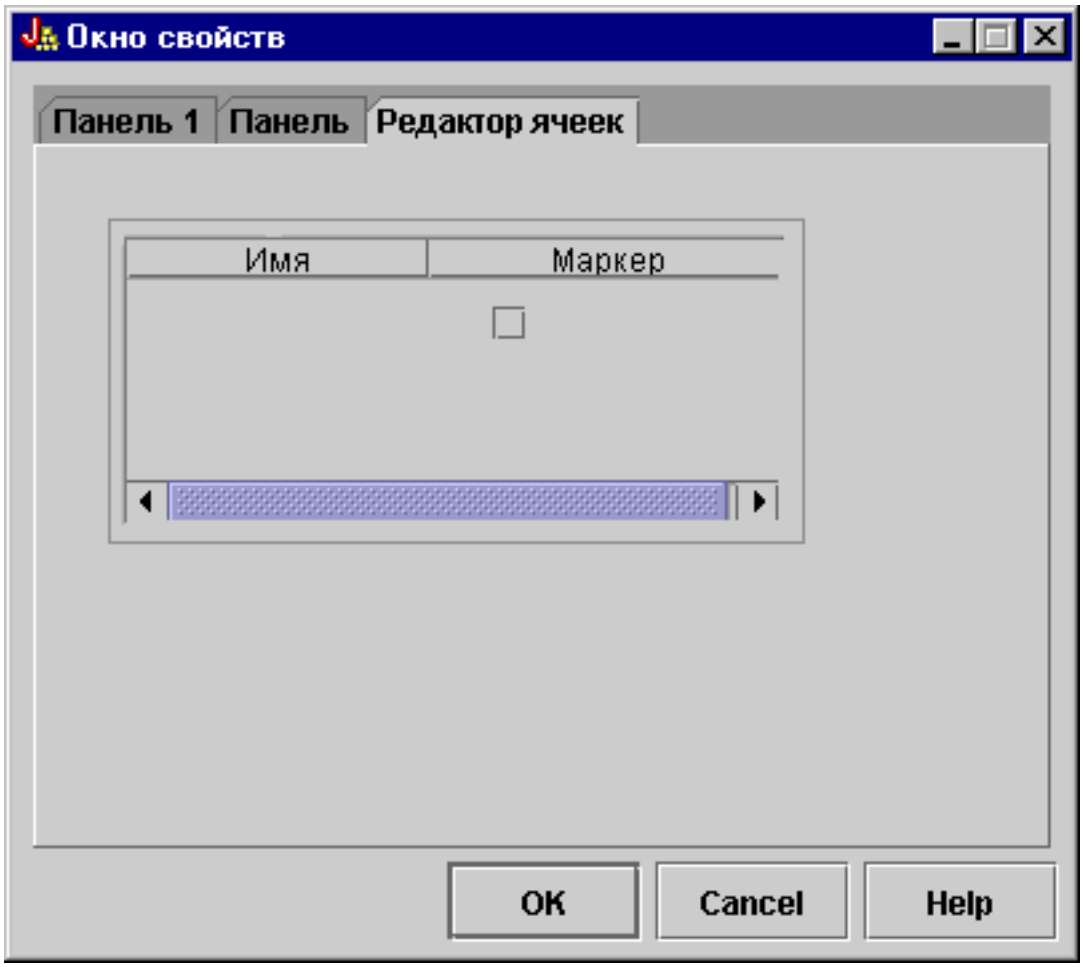


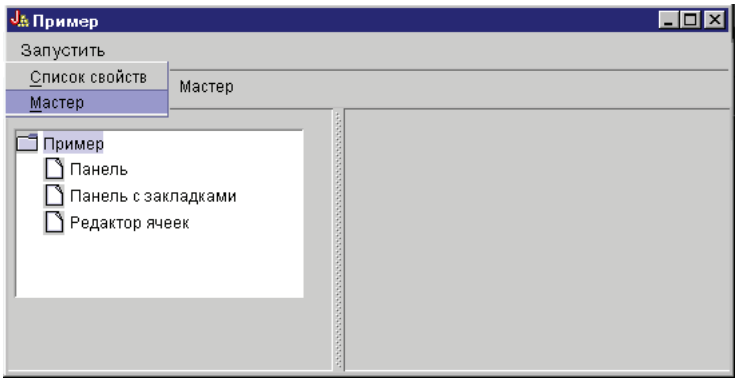
Рисунок 7: вкладка Редактор таблицы



Запуск мастера

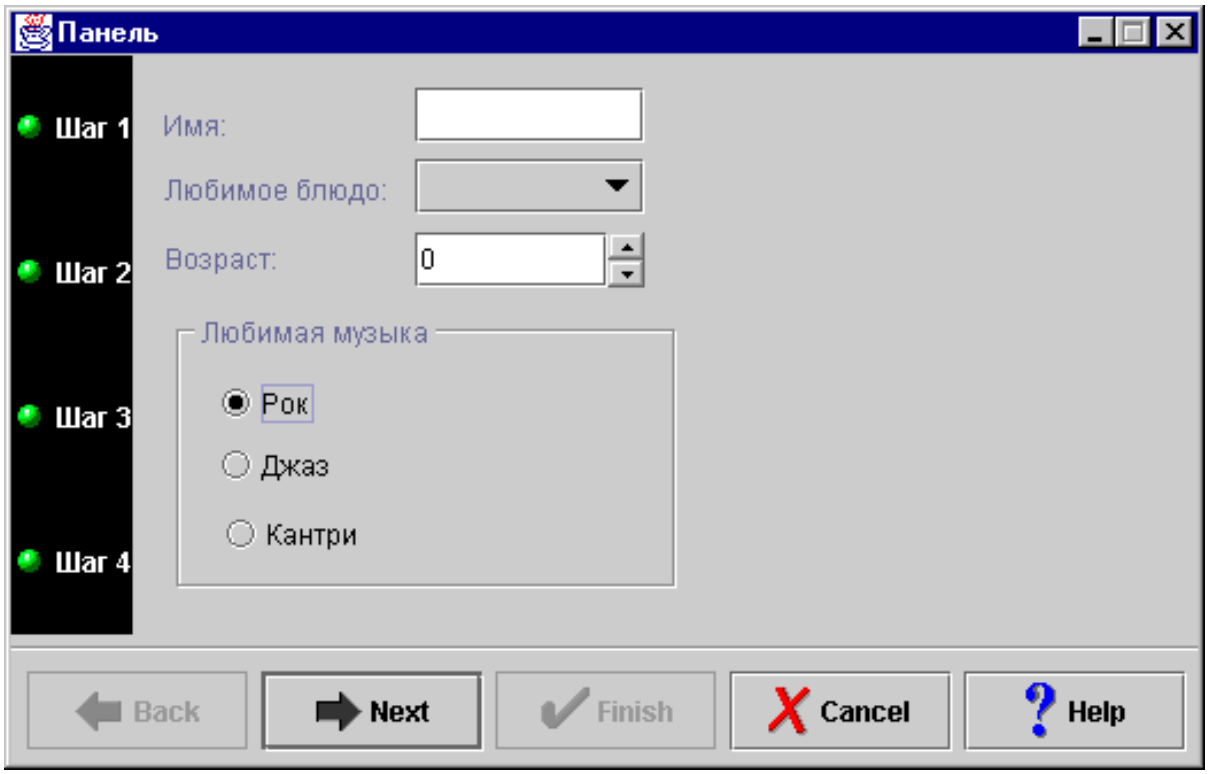
Мастер можно запустить с помощью кнопки на панели инструментов или из меню **Запустить**. В этом примере демонстрируется связь между пунктами меню и элементами панели инструментов. На рис. 8 показан пункт **Мастер** меню **Запустить** главного окна GUI Builder.

Рисунок 8: выбор пункта Мастер в меню Запустить



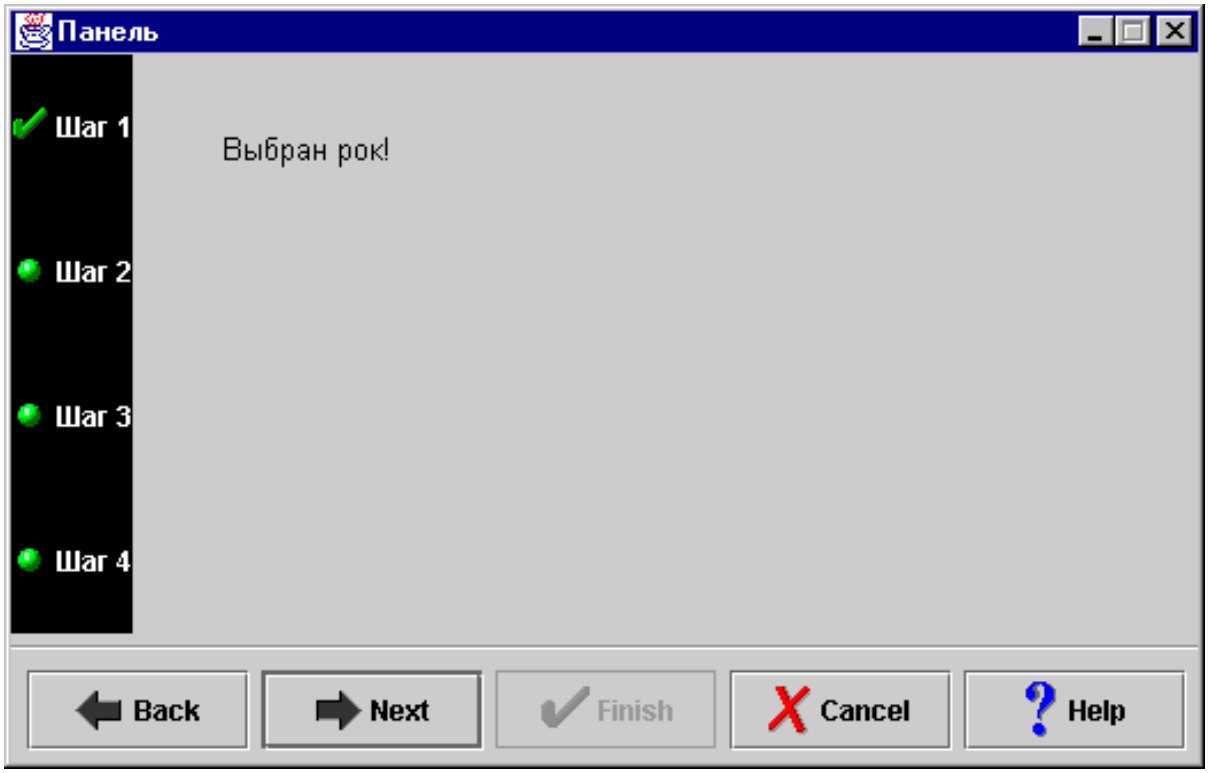
На рис. 9 показаны опции первого окна мастера.

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

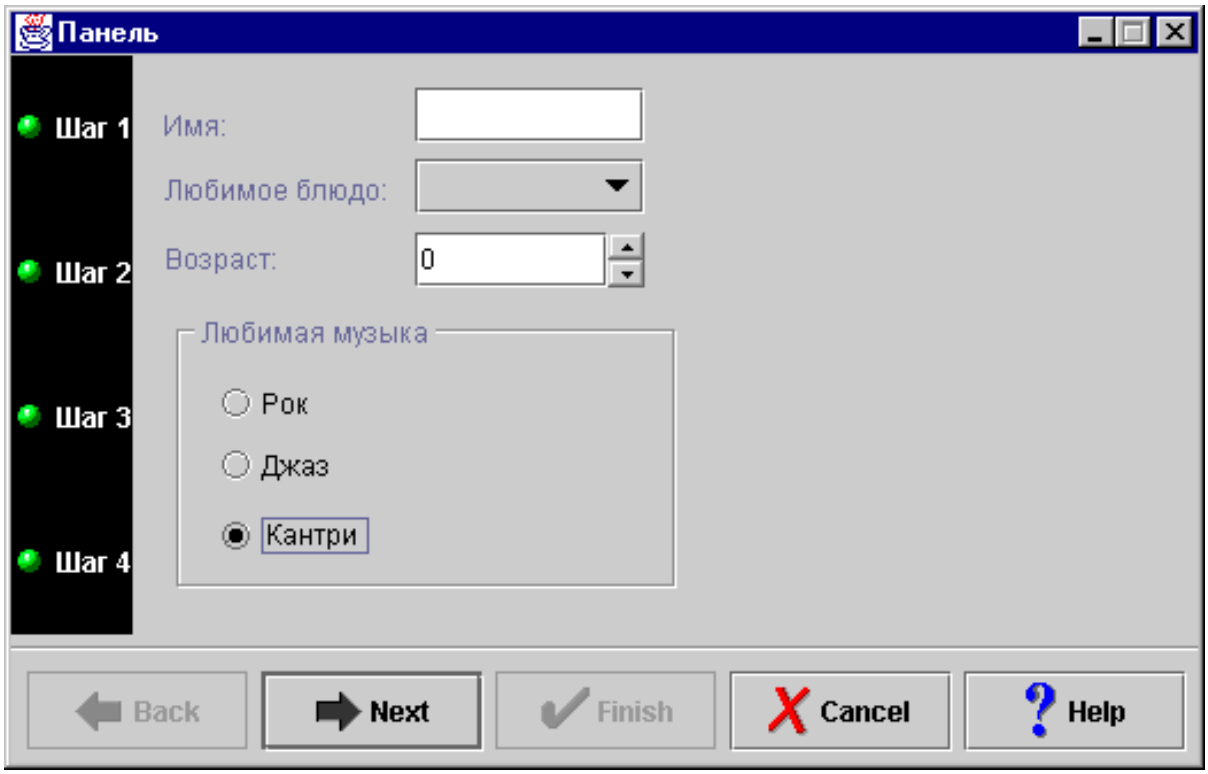


Рисунок 13: второе окно мастера (после выбора опции Кантри)

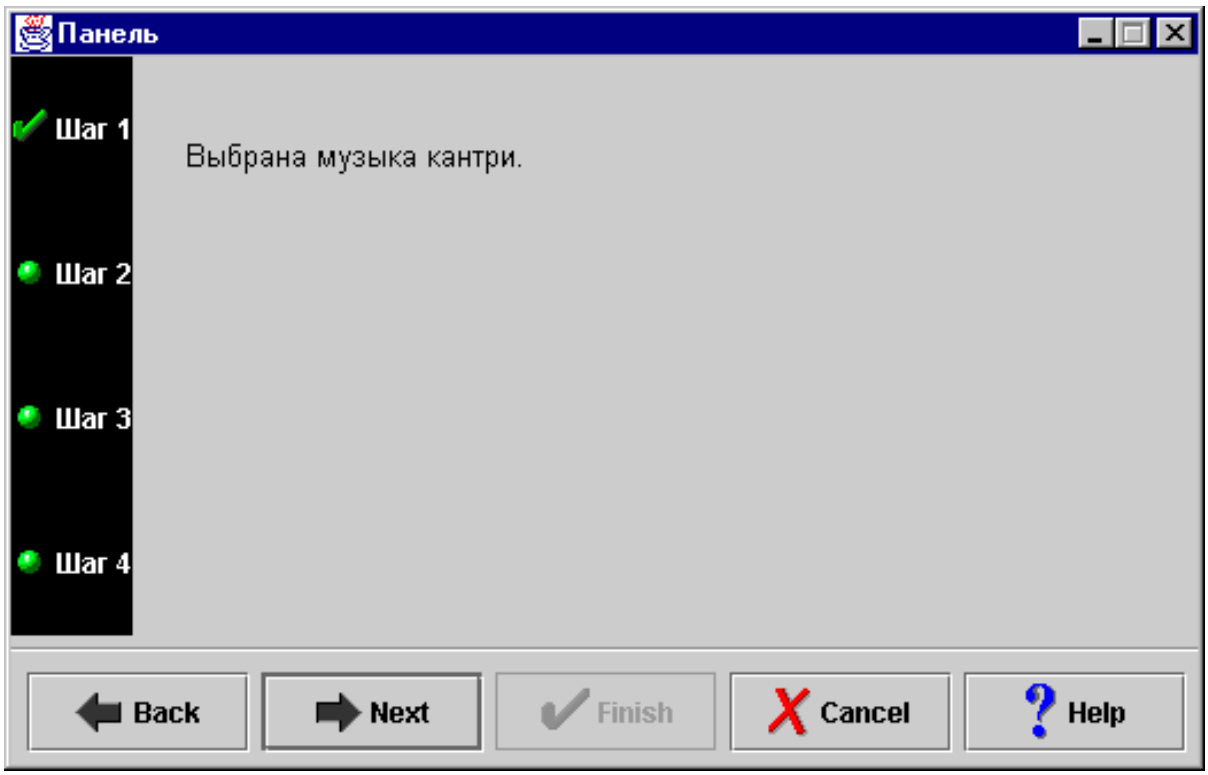
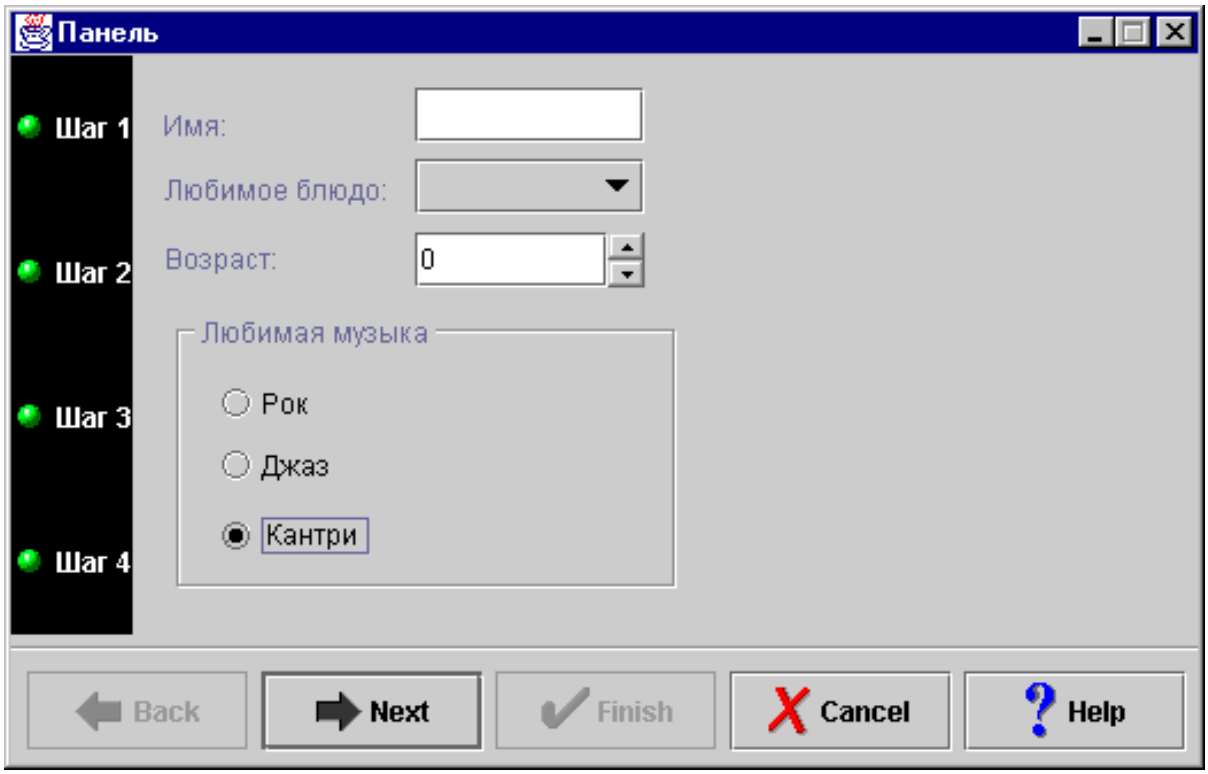


Рисунок 14: Возврат в первое окно мастера

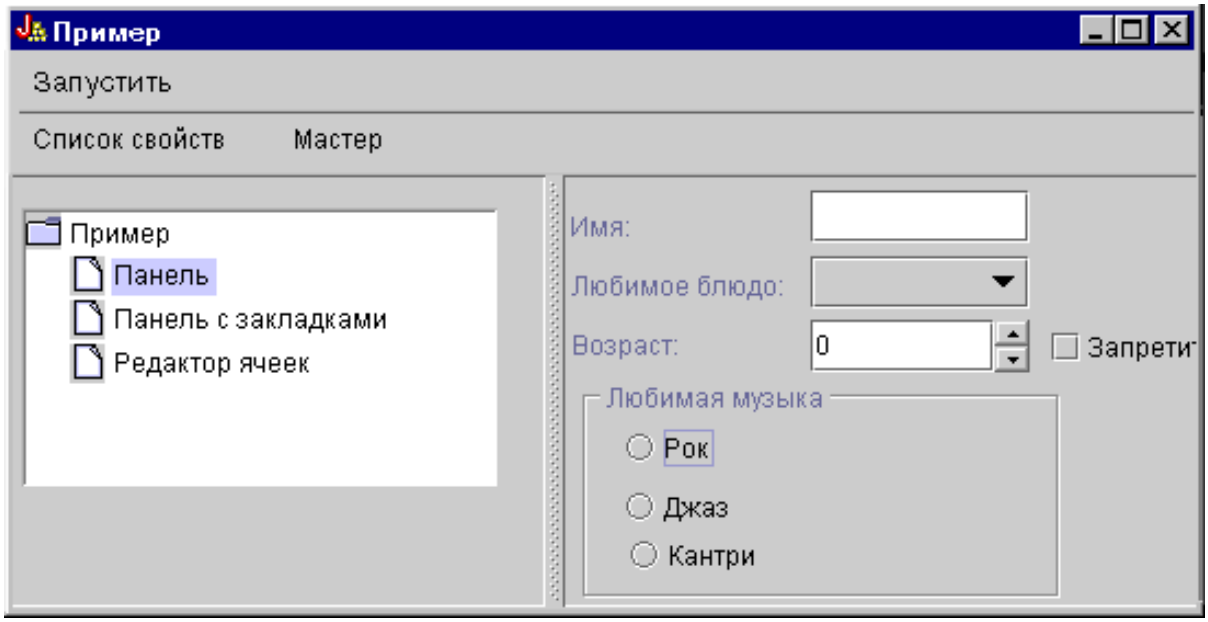


Это означает, что программист запретил выбирать кантри в качестве любимого стиля музыки.

Просмотр примеров

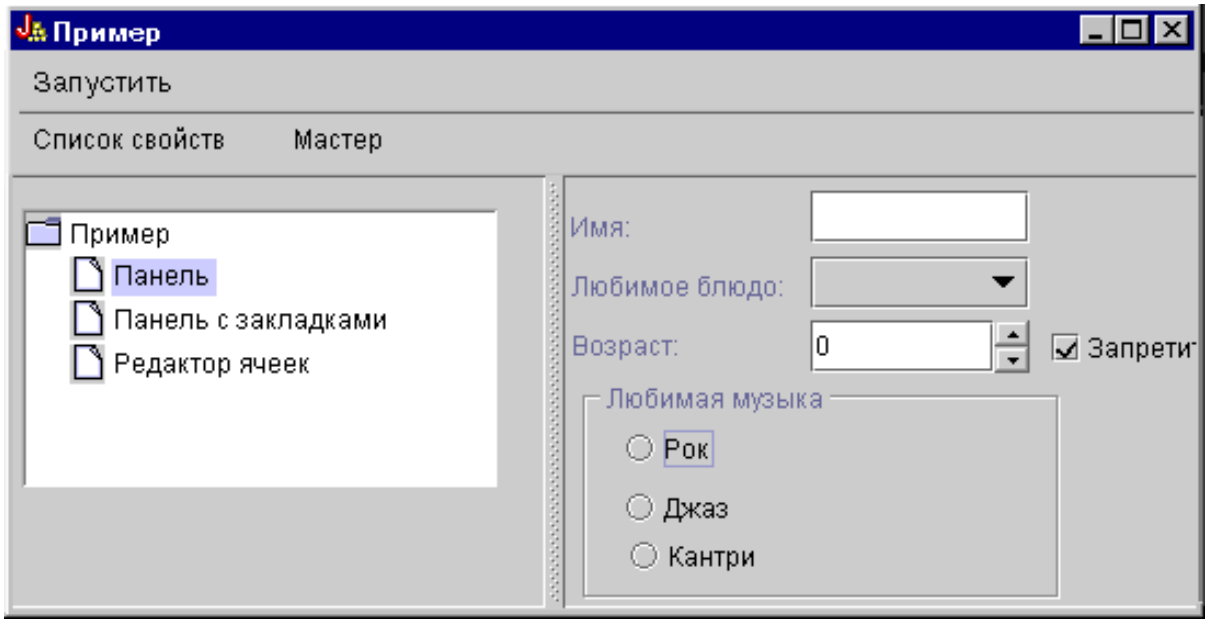
В левой панели главного окна примера можно выбрать и другие функции. На рис. 15 показано окно, которое появится после выбора пункта **Панель** в левой панели окна.

Рисунок 15: выбор ощии Панель в левой панели окна



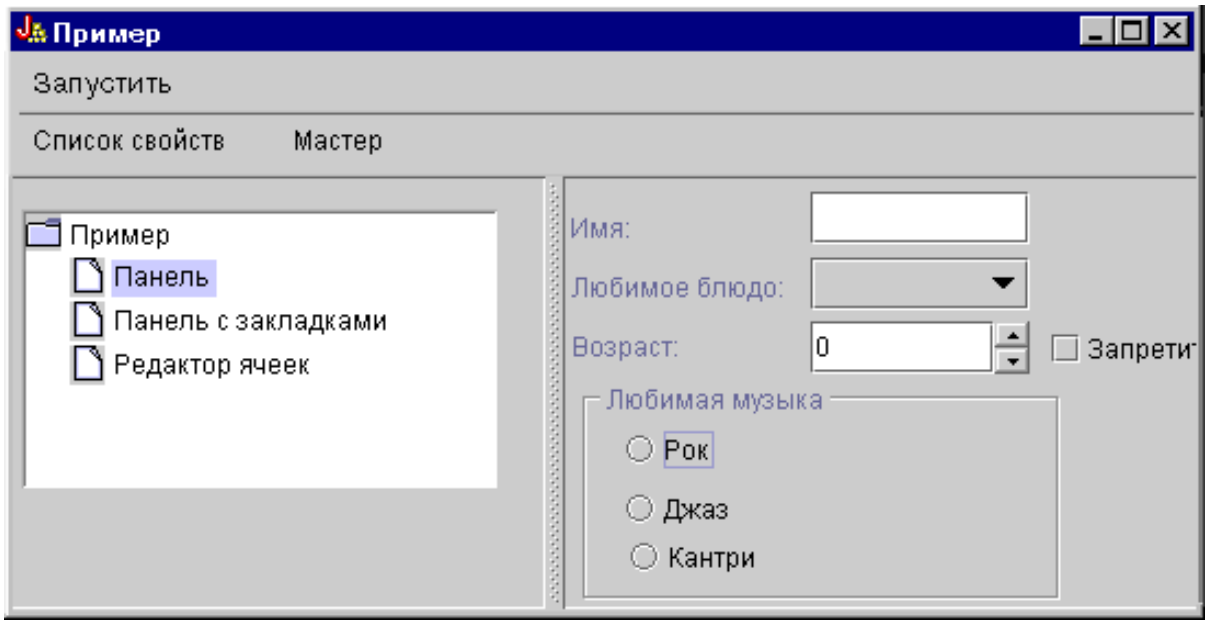
В этом примере предусмотрена опция, отключающая вывод изображений. Если вы выберете опцию **Отключить вывод изображений**, то изображения не будут показаны в окне (см. рис. 16).

Рисунок 16: выбор опции Отключить вывод изображений в правой панели



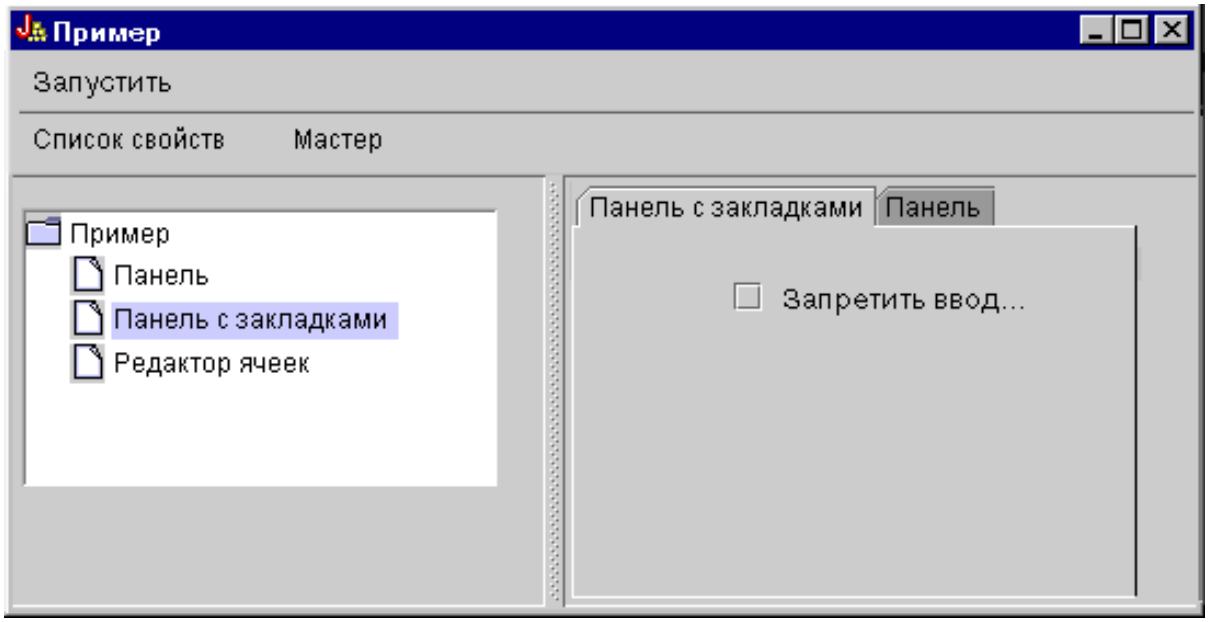
Данный пример также демонстрирует выпадающий список (см. рис. 17).

Рисунок 17: Выбор элемента в списке Любимое блюдо в правой панели



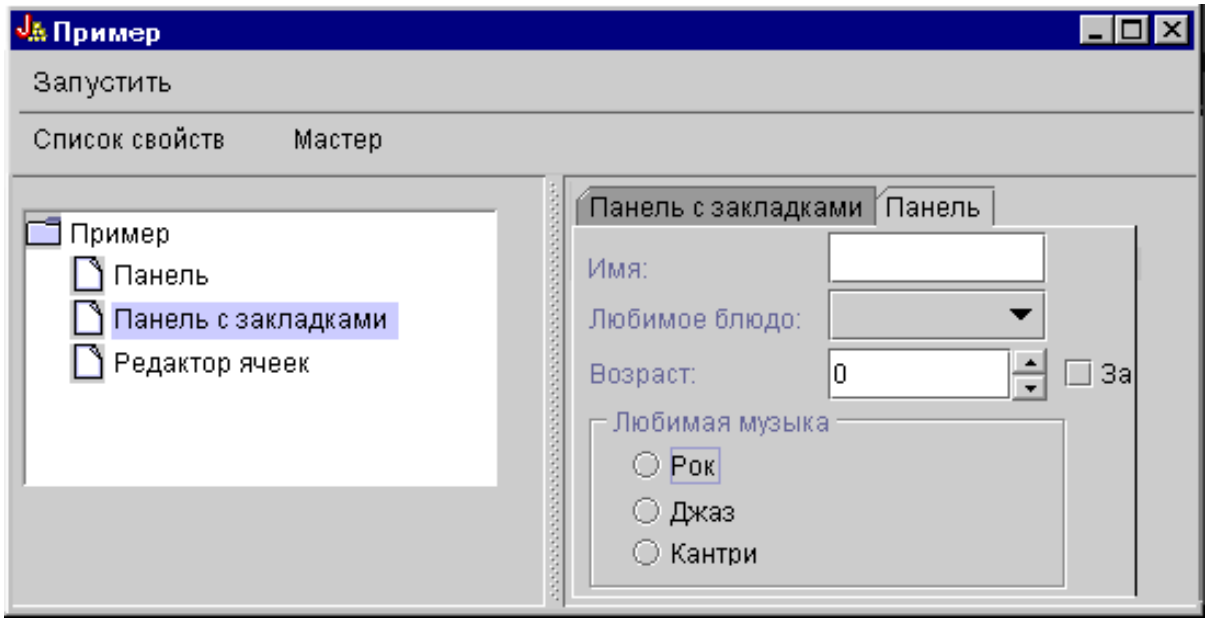
На рис. 18 показано окно, появляющееся после выбора опции **Панель с закладками** в левой панели примера.

Рисунок 18: выбор опции Панель с закладками в левой панели



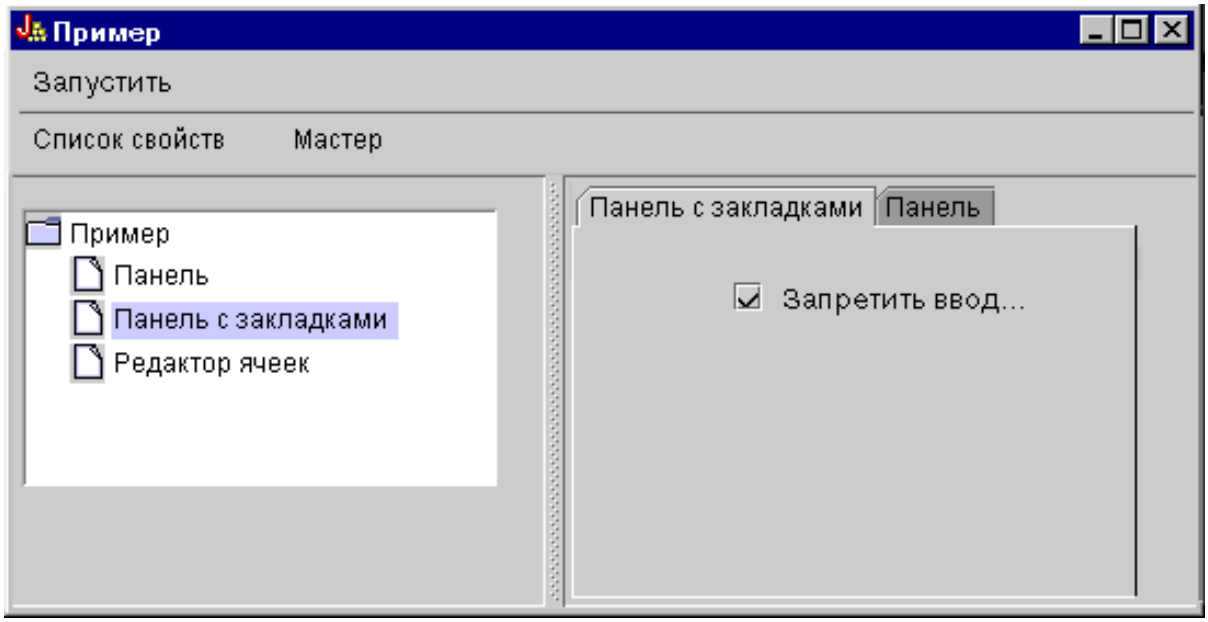
На рис. 19 показано окно, появляющееся после выбора вкладки **Пример панели** в правой панели.

Рисунок 19: вкладка Пример панели в правой панели



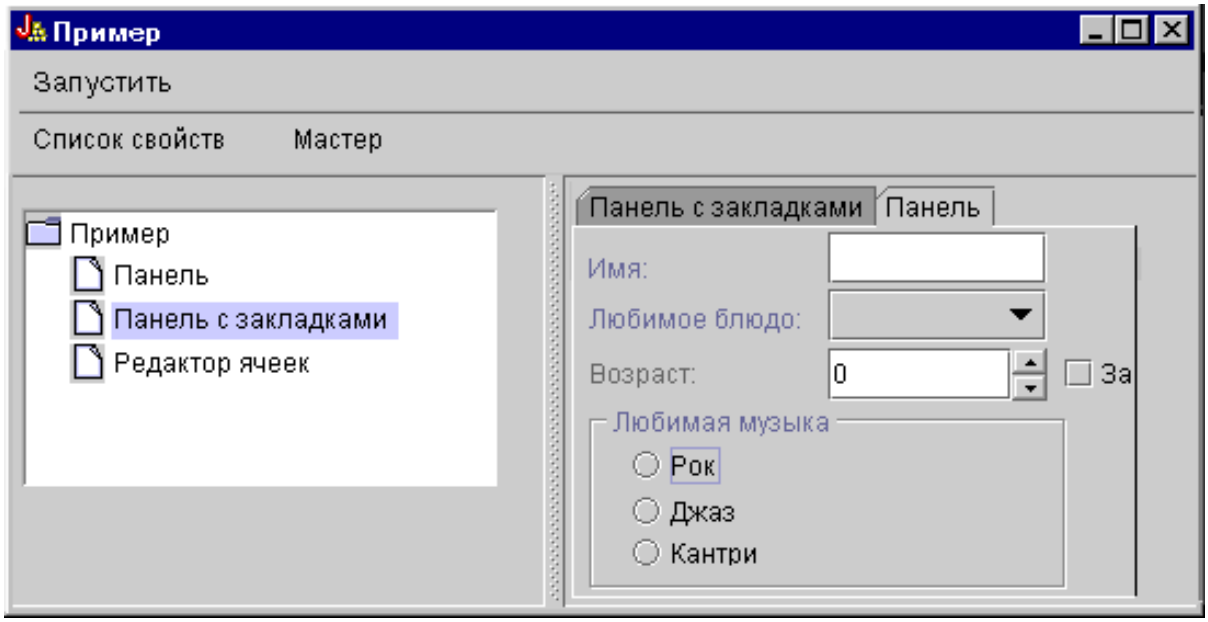
Вновь выберите **Вкладку 1** (в правой панели), затем щелкните в поле **Запретить ввод возраста на вкладке 2**.

Рисунок 20: Выбор опции Запретить ввод возраста на вкладке 2 в правой панели



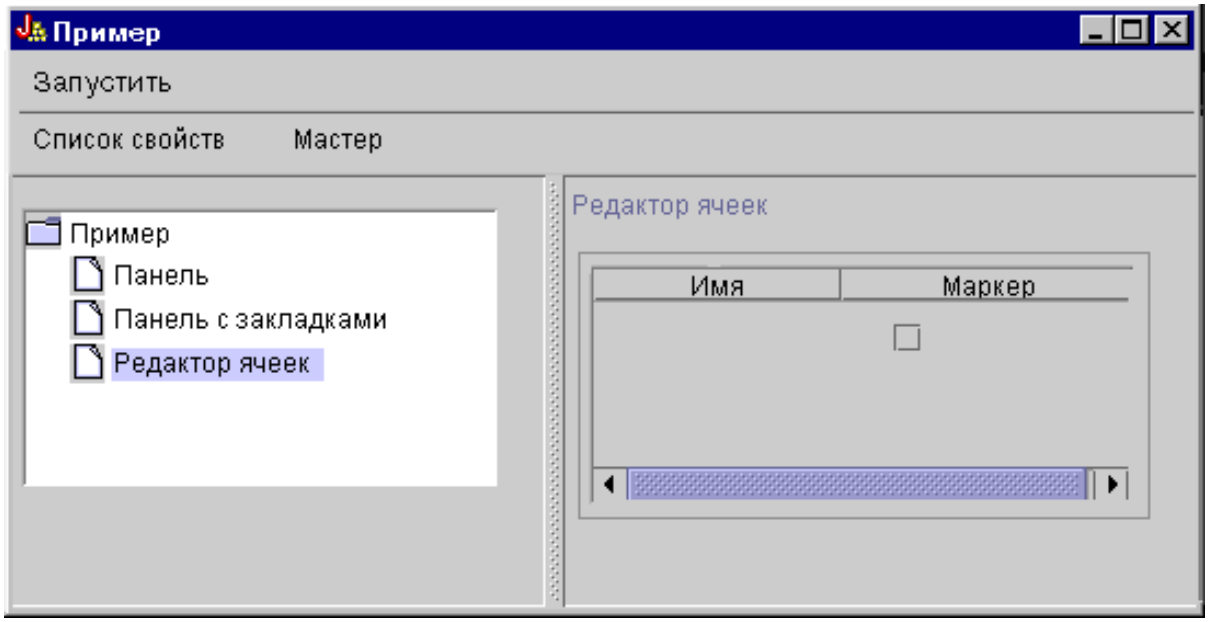
После выбора опции **Запретить ввод возраста на вкладке 2** поле **Возраст** на вкладке **Пример панели** станет недоступным (см. рис. 21).

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

Некоторые возможности применения классов HTML продемонстрированы в следующих примерах:

- Пример: Применение класса BidiOrdering
- 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.

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> **Отказ от гарантий на предоставляемый код**
>
> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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

Ниже приведен пример применения классов форм HTML, 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.

```
////////////////////////////////////////////////////////////////////////
//
// В этом примере демонстрируется применение классов HTML из
// AS/400 Toolbox для Java для создания форм HTML.
//
////////////////////////////////////////////////////////////////////////

package customer;
```

```
import java.io.*;
import java.awt.Color;

import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.as400.access.*;
import com.ibm.as400.util.html.*;


public class HTMLExample extends HttpServlet
{
  // Определение того, находится ли пользователь в списке регистрантов.
  private static boolean found = false;

  // Файл для хранения регистрационных данных
  String regPath = "c:\\registration.txt";


  public void init(ServletConfig config)
  {

    try
    {
       super.init(config);

    }
    catch(Exception e)
    {
       e.printStackTrace();
    }

  }

  /**
   *  Обработка запроса GET.
   *  В параметре req - запрос.
   *  В параметре res - ответ.
   **/
   public void doGet (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
   {
      res.setContentType("text/html");
      ServletOutputStream out = res.getOutputStream();

      // Получение исходного текста страницы из класса HTML
      out.println(showHTML());
      out.close();
   }


    public void doPost (HttpServletRequest req, HttpServletResponse res)
       throws ServletException, IOException
    {
        String nameStr   = req.getParameter("name");
        String emailStr = req.getParameter("email");
        String errorText= "";

        // Поток передачи данных сервлету
        ServletOutputStream out = res.getOutputStream();

        res.setContentType("text/html");

        // Проверка имени и электронного адреса клиента
        if (nameStr.length() == 0)
           errorText += "Не указано имя пользователя. ";
```

```
if (emailStr.length() == 0)
   errorText += "Не указан электронный адрес. " ;

// Если имя и электронный адрес указаны, то - продолжение
if (errorText.length() == 0)
{

   try
   {
      // Создание файла registration.txt
      FileWriter f = new FileWriter(regPath, true);
      BufferedWriter output = new BufferedWriter(f);

      // Буферизованное чтение для поиска в файле
      BufferedReader in = new BufferedReader(new FileReader(regPath));

      String line = in.readLine();

      // Сброс флага found
      found = false;

      // Проверка, не зарегистрирован ли уже клиент
      // с тем же именем или электронным адресом
      while (!found)
      {
         // если файл пуст или просмотрен полностью
         if (line == null)
            break;

         // если клиент уже зарегистрирован
         if ((line.equals("Имя клиента: " + nameStr)) ||
            (line.equals("Электронный адрес: " + emailStr)))
         {
            // Вывод сообщения с информацией о том,
            // что он уже зарегистрирован
            out.println ("<HTML> " +
                         "<TITLE> Toolbox Registration</TITLE> " +
                         "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> " +
                         "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> " );
            out.println ("<P><HR>" +
                         "<P>" + nameStr +
                         "</B>, вы уже зарегистрированы с этим " +
                         "<B>именем</B> или <B>адресом электронной почты</B>." +
                         "<P> Благодарим вас!...<P><HR>");

            // Создание и вывод объекта HTMLHyperlink
            out.println ("<UL><LI>" +
                         new HTMLHyperlink("./customer.HTMLExample",
                                           "Вернуться к форме регистрации") +
                         "</UL></BODY></HTML>");
            found = true;
            break;

         }
         else    // чтение следующей строки
            line = in.readLine();

      }

      // Объект String для хранения данных, полученных из формы HTML
      String data;

      // Если имени или электронного адреса пользователя нет
      // в списке, продолжить выполнение операции.
      if (!found)
      {
         //-------------------------------------------------------------
```

```
// Добавление данных о новом клиенте в файл
output.newLine();
output.write("Имя клиента: " + nameStr);
output.newLine();
output.write("Электронный адрес: " + emailStr);
output.newLine();
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значения переключателя "USE" формы
data = req.getParameter("use");
if(data != null)
{
   output.write("Работает с AS/400 Toolbox для Java: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значения переключателя "Требуется дополнительная информация"
data = req.getParameter("contact");
if (data != null)
{
   output.write("Требуется дополнительная информация: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значения поля "Версия AS400" формы
data = req.getParameter("version");
if (data != null)
{
   if (data.equals("multiple versions"))
   {
      data = req.getParameter("MultiList");
      output.write("Применяемые версии: " + data);
   }
   else
      output.write("Версия AS400: " + data);

   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значения поля "Область применения Java" формы
data = req.getParameter("interest");
if (data != null)
{
   output.write("Область текущего или будущего применения Java: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значения поля "Платформы"
data = req.getParameter("platform");
if (data != null)
{
   output.write("Платформы: " + data);
   output.newLine();
```

```
   if (data.indexOf("Other") >= 0)
   {
      output.write("Другие платформы: " + req.getParameter("OtherPlatforms"));
      output.newLine();
   }
}
//-------------------------------------------------------------


//-------------------------------------------------------------
//Получение из формы значения "Число серверов iSeries"
data = req.getParameter("list1");
if (data != null)
{
   output.write("Число серверов iSeries: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значения поля "Комментарии" формы
data = req.getParameter("comments");
if (data != null && data.length() > 0)
{
   output.write("Комментарии: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значение поля "Вложение"
data = req.getParameter("myAttachment");
if (data != null && data.length() > 0)
{
   output.write("Вложение: " + data);
   output.newLine();
}
//-------------------------------------------------------------


//-------------------------------------------------------------
// Получение значения скрытого поля "Copyright"
data = req.getParameter("copyright");
if (data != null)
{
   output.write(data);
   output.newLine();
}
//-------------------------------------------------------------


output.flush();
output.close();

// Выдача сообщения "Спасибо!"
out.println("<HTML>");
out.println("<TITLE>Благодарим вас!</TITLE>");
out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
out.println("<BODY BGCOLOR=\"blanchedalmond\">");
out.println("<HR><P>Благодарим вас за регистрацию, <B>" + nameStr + "</B>!<P><HR>");

// Создание и вывод объекта HTMLHyperlink
out.println("<UL><LI>" +
            new HTMLHyperlink("./customer.HTMLExample",
                "Вернуться к форме регистрации"));
```

```
            out.println("</UL></BODY></HTML>");

          }

        }
       catch (Exception e)
       {
          // Вывод сообщение об ошибке в браузере
          out.println("<HTML>");
          out.println("<TITLE>ОШИБКА!</TITLE>");
          out.println("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> ");
          out.println("<BODY BGCOLOR=\"blanchedalmond\">");
          out.println("<BR><B>Сообщение об ошибке:</B><P>");
          out.println(e + "<P>");

          // Создание и вывод объекта HTMLHyperlink
          out.println("<UL><LI>" +
                      new HTMLHyperlink("./customer.HTMLExample", "Вернуться к форме регистрации"));
          out.println("</UL></BODY></HTML>");

          e.printStackTrace();
       }
    }
    else
    {
       // Вывод сообщения о том, что имя пользователя и электронный
       // адрес не указаны, с предложением повторить ввод
       out.println ("<HTML> " +
                    "<TITLE>Неверные данные в форме регистрации</TITLE> " +
                    "<META HTTP-EQUIV=\"pragma\" content=\"no-cache\"> " +
                    "<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"> " );

       out.println ("<HR><B>ОШИБКА</B> в предоставленных заказчиком данных - <P><B>" +
                    errorText +
                    "</B><P>Пожалуйста, повторите ввод... <HR>");


       // Создание и вывод объекта HTMLHyperlink
       out.println("<UL><LI>" +
                   new HTMLHyperlink("./customer.HTMLExample", "Вернуться к форме регистрации"));
                   "</UL></BODY></HTML>");
    }
     // Закрытие потока
     out.close();


}


public void destroy(ServletConfig config)
{
   // никаких действий выполнять не нужно
}


public String getServletInfo()
{
    return "Регистрация продукта";
}

private String showHTML()
{
  // Буфер для хранения Web-страницы
  StringBuffer page = new StringBuffer();

  // Создание объекта формы HTML.
  HTMLForm form = new HTMLForm("/servlet/customer.HTMLExample");;
```

```
HTMLText txt;

// Создание заголовка Web-страницы и копирование его в буфер
page.append("<HTML>\n");
page.append("<TITLE> Добро пожаловать!!</TITLE>\n");
page.append("<HEAD><SCRIPT LANGUAGE=\"JavaScript\">
            function test(){alert(\"Это пример сценария, исполняемого с помощью
            ButtonFormInput.\")}</SCRIPT></HEAD>");
page.append("<META HTTP-EQUIV=\"pragma\" content=\"no-cache\">\n");
page.append("<BODY BGCOLOR=\"blanchedalmond\" TEXT=\"black\"><BR>\n");

try
{
   //-----------------------------------------------------------------
   // Создание заголовка страницы с помощью класса HTMLText
    txt = new HTMLText("Регистрация продукта");
   txt.setSize(5);
    txt.setBold(true);
   txt.setColor(new Color(199, 21, 133));
   txt.setAlignment(HTMLConstants.CENTER);

   // Вывод текста HTML в буфер
   page.append(txt.getTag(true) + "<HR><BR>\n");
   //-----------------------------------------------------------------

   //-----------------------------------------------------------------
   // Создание разметки строк для имени и электронного адреса
   LineLayoutFormPanel line = new LineLayoutFormPanel();
    txt = new HTMLText("Введите свое имя и электронный адрес:");
   txt.setSize(4);
   line.addElement(txt);

   // Вывод формы в буфер
   page.append(line.toString());
   page.append("<BR>");
   //-----------------------------------------------------------------

   //-----------------------------------------------------------------
   // Выбор метода для формы
   form.setMethod(HTMLForm.METHOD_POST);
   //-----------------------------------------------------------------

   //-----------------------------------------------------------------
   // Создание поля для ввода имени.
   TextFormInput user = new TextFormInput("name");
   user.setSize(25);
   user.setMaxLength(40);

   // Создание поля для ввода электронного адреса.
   TextFormInput email = new TextFormInput("email");
   email.setSize(30);
   email.setMaxLength(40);

   // Создание объекта ImageFormInput
   ImageFormInput img = new ImageFormInput("Отправить форму", "..\\images\\myPiimages/c.gif");
   img.setAlignment(HTMLConstants.RIGHT);
   //-----------------------------------------------------------------

   //-----------------------------------------------------------------
   // Создание объекта LineLayoutFormPanel для имени и электронного адреса
   LineLayoutFormPanel line2 = new LineLayoutFormPanel();

   // Добавление к форму элементов для ввода имени
            line2.addElement(new LabelFormElement("Имя:"));
   line2.addElement(user);
   // Добавление к форме элементов для ввода электронного адреса
   line2.addElement(new LabelFormElement("Электронный адрес:"));
```

```
line2.addElement(email);
line2.addElement(img);
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Создание разметки строк для вопросов
LineLayoutFormPanel line3 = new LineLayoutFormPanel();

// Добавление к форме элементов
line3.addElement(new LineLayoutFormPanel());
line3.addElement(new CheckboxFormInput("use", "yes",
      "Работаете ли вы с AS/400 Toolbox для Java?", false));
line3.addElement(new LineLayoutFormPanel());
line3.addElement(new CheckboxFormInput("contact", "yes",
      "Требуется ли вам информация о будущих выпусках этого
      продукта?", true));
line3.addElement(new LineLayoutFormPanel());
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Создание группы радиокнопок для выбора версии
RadioFormInputGroup group = new RadioFormInputGroup("version");

// Добавление различных вариантов в эту группу
group.add(new RadioFormInput("version", "v3r2", "V3R2", false));
group.add(new RadioFormInput("version", "v4r1", "V4R1", false));
group.add(new RadioFormInput("version", "v4r2", "V4R2", false));
group.add(new RadioFormInput("version", "v4r3", "V4R3", false));
group.add(new RadioFormInput("version", "v4r4", "V4R4", false));
group.add(new RadioFormInput("version", "multiple versions", "Различные версии? Какие:", false));

// Создание элемента, допускающего выбор нескольких значений
SelectFormElement mlist = new SelectFormElement("MultiList");
mlist.setMultiple(true);
mlist.setSize(3);

// Добавление нескольких вариантов к предыдущему элементу
SelectOption option1 = mlist.addOption("V3R2", "v3r2");
SelectOption option2 = mlist.addOption("V4R1", "v4r1");
SelectOption option3 = mlist.addOption("V4R2", "v4r2");
SelectOption option4 = mlist.addOption("V4R3", "v4r3");
SelectOption option5 = mlist.addOption("V4R4", "v4r4");

// Создание текста HTML
txt = new HTMLText("Текущий уровень сервера:");
txt.setSize(4);

// Создание табличной разметки
GridLayoutFormPanel grid1 = new GridLayoutFormPanel(3);

// Добавление к ней группы радиокнопок и переключателей
grid1.addElement(txt);
grid1.addElement(group);
grid1.addElement(mlist);
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Создание табличной разметки для области применения
GridLayoutFormPanel grid2 = new GridLayoutFormPanel(1);
txt = new HTMLText("Текущие проекты или область интересов: " +
                   "(проверка всех возможных)");
txt.setSize(4);

// Добавление элементов к табличной разметке
grid2.addElement(new LineLayoutFormPanel());
grid2.addElement(txt);
// Создание и добавление переключателя к табличной разметке
```

```
grid2.addElement(new CheckboxFormInput("interest", "applications", "Приложения", true));
grid2.addElement(new CheckboxFormInput("interest", "applets", "Аплеты", false));
grid2.addElement(new CheckboxFormInput("interest", "servlets", "Сервлеты", false));
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Создание разметки строк для платформ
LineLayoutFormPanel line4 = new LineLayoutFormPanel();
txt = new HTMLText("Использованные клиентские платформы: " +
                   "(проверка всех возможных)");
txt.setSize(4);

// Добавление элементов к разметке строк
line4.addElement(new LineLayoutFormPanel());
line4.addElement(txt);
line4.addElement(new LineLayoutFormPanel());
line4.addElement(new CheckboxFormInput("platform", "95", "Windows95", false));
line4.addElement(new CheckboxFormInput("platform", "98", "Windows98", false));
line4.addElement(new CheckboxFormInput("platform", "NT", "WindowsNT", false));
line4.addElement(new CheckboxFormInput("platform", "OS2", "OS/2", false));
line4.addElement(new CheckboxFormInput("platform", "AIX", "AIX", false));
line4.addElement(new CheckboxFormInput("platform", "Linux", "Linux", false));
line4.addElement(new CheckboxFormInput("platform", "AS400", "iSeries", false));
line4.addElement(new CheckboxFormInput("platform", "Other", "Другие:", false));

TextFormInput other = new TextFormInput("OtherPlatforms");
other.setSize(20);
other.setMaxLength(50);

line4.addElement(other);
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Создание поля для числа серверов
LineLayoutFormPanel grid3 = new LineLayoutFormPanel();

txt = new HTMLText(
   "Сколько серверов iSeries вы используете?");
txt.setSize(4);

// Создание элемента для выбора числа серверов
SelectFormElement list = new SelectFormElement("list1");
// Создание и добавление к этому элементу различных вариантов
SelectOption opt0 = list.addOption("0", "нет");
SelectOption opt1 = list.addOption("1", "одна", true);
SelectOption opt2 = list.addOption("2", "две");
SelectOption opt3 = list.addOption("3", "три");
SelectOption opt4 = list.addOption("4", "четыре");
SelectOption opt5 = new SelectOption("5+", "пять или более", false);
list.addOption(opt5);

// Добавление элементов к табличной разметке
grid3.addElement(new LineLayoutFormPanel());
grid3.addElement(txt);
grid3.addElement(list);
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Создание табличной разметки для комментариев к продукту
GridLayoutFormPanel grid4 = new GridLayoutFormPanel(1);
 txt = new HTMLText("Комментарии к продукту:");
txt.setSize(4);

// Добавление элементов к табличной разметке
grid4.addElement(new LineLayoutFormPanel());
grid4.addElement(txt);
// Создание поля ввода для комментариев
```

```
   grid4.addElement(new TextAreaFormElement("comments", 5, 75));
   grid4.addElement(new LineLayoutFormPanel());
   //-----------------------------------------------------------------

   //-----------------------------------------------------------------
   // Создание табличной разметки
   GridLayoutFormPanel grid5 = new GridLayoutFormPanel(2);
   txt = new HTMLText("Войти в систему?");
   txt.setSize(4);

   // Создание поля ввода и метки для имени системы.
   TextFormInput sys = new TextFormInput("system");
   LabelFormElement sysLabel = new LabelFormElement("Система:");

   // Создание поля ввода и метки для имени пользователя.
   TextFormInput uid = new TextFormInput("uid");
   LabelFormElement uidLabel = new LabelFormElement("Имя пользователя");

   // Создание поля ввода пароля и метки для пароля.
   PasswordFormInput pwd = new PasswordFormInput("pwd");
   LabelFormElement pwdLabel = new LabelFormElement("Пароль");

   // Добавление созданных элементов к табличной разметке
   grid5.addElement(sysLabel);
   grid5.addElement(sys);
   grid5.addElement(uidLabel);
   grid5.addElement(uid);
   grid5.addElement(pwdLabel);
   grid5.addElement(pwd);
   //-----------------------------------------------------------------

   //-----------------------------------------------------------------
   // Добавление в форму HTML различных созданных
   // панелей в порядке, в котором они будут появляться
    form.addElement(line2);
   form.addElement(line3);
   form.addElement(grid1);
   form.addElement(grid2);
   form.addElement(line4);
   form.addElement(grid3);
   form.addElement(grid4);
   form.addElement(txt);
   form.addElement(new LineLayoutFormPanel());
   form.addElement(grid5);
   form.addElement(new LineLayoutFormPanel());
   form.addElement(
      new HTMLText("Выберите вложение: <br />"));
   // Добавление поля выбора файла к форме
   form.addElement(new FileFormInput("myAttachment"));
   form.addElement(new ButtonFormInput("button", "Тест", "test()"));
   // Добавление пустой разметки строк к форме. Это приведет
   // к появлению символа конца строки <br /> в форме
   form.addElement(new LineLayoutFormPanel());
   form.addElement(new LineLayoutFormPanel());
   form.addElement(new SubmitFormInput("submit", "Зарегистрироваться"));
   form.addElement(new LineLayoutFormPanel());
   form.addElement(new LineLayoutFormPanel());
   form.addElement(new ResetFormInput("reset", "Сброс"));
   // Добавление скрытого поля к форме
   form.addElement(new HiddenFormInput("copyright", "(C) Copyright IBM Corp. 1999, 1999"));
   //-----------------------------------------------------------------

   // Вывод всей формы в буфер
   page.append(form.toString());

}
catch(Exception e)
```

```
      {
         e.printStackTrace();
      }

      // Вывод закрывающих тегов HTML в буфер
      page.append("</BODY>\n");
      page.append("</HTML>\n");

      // Выдача созданной страницы
      return page.toString();

   }
}
```

Пример вывода класса HTML

Ниже приведены примеры вывода, полученного в результате применения примера класса HTML:

- Имя клиента: дядя Федор
  Электронный адрес: fedor@prostokvashino.ru
  Работает с AS/400 Toolbox для Java: да
  Требуется дополнительная информация: да
  Применяемые версии: v4r2,v4r4
  Область текущего или будущего применения Java: приложения,сервлеты
  Платформы: NT,Linux
  Число серверов iSeries: три
  Комментарии: Toolbox применяется всем отделом разработки для создания приложений по заказам пользователей
  Вложение: U:\wiedrich\servlet\temp.html
  (C) Copyright IBM Corp. 1999, 1999
- Имя клиента: Карлсон
  Электронный адрес: carlson@roof.sv
  Работает с AS/400 Toolbox для Java: да
  Версия AS400: v4r4
  Область текущего или будущего применения Java: сервлеты
  Платформы: OS2
  Число серверов iSeries: пять или более
  (C) Copyright IBM Corp. 1999, 1999
- Имя клиента: Винни-Пух
  Электронный адрес: pooh@home.com
  Требуется дополнительная информация: да
  Версия AS400: v4r2
  Область текущего или будущего применения Java: приложения
  Платформы: NT,Другие
  Другие платформы: Solaris
  Число серверов iSeries: один
  Комментарии: Это первая программа, в которой используется этот класс! Очень хорошо!
  (C) Copyright IBM Corp. 1999, 1999
- Имя клиента: Незнайка
  Электронный адрес: neznayka@suncity.su
  Версия AS400: v4r2
  Число серверов iSeries: один
  (C) Copyright IBM Corp. 1999, 1999

Пример: Применение классов HTMLTree

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)

Приведенный фрагмент кода в сочетании с кодом из двух других файлов демонстрирует 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 - 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 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

Ниже приведен пример работы с классами HTMLTable:

```
     // 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>Баланс счета заказчика - 1 января 2000 г.</caption>
<tr>
<th>Счет</th>
<th>Имя</th>
<th>Баланс</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 2. 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 OS/400. После описания каждого примера приведена ссылка на документ, содержащий исходный код PCML и программу на Java.

- Простой пример получения данных: Содержит исходный код PCML и программу на Java, которая предназначена для получения информации о пользовательском профайле сервера. В этом примере вызывается API *Получить информацию о пользователе* (**QSYRUSRI**).
- Получение списка пользователей: Содержит исходный код PCML и программу на Java, предназначенную для получения списка пользователей, которым разрешен доступ к серверу. В ней вызывается API *Открыть список пользователей с правами доступа* (**QGYOLAUS**). Пример иллюстрирует работу с массивом структур, полученным от программы сервера.
- Получение многомерных данных: Содержит исходный код PCML и программу на Java, предназначенную для получения списка файлов NFS, экспортируемых из сервера. В этом примере вызывается API *Получить список экспортируемых файлов NFS* (**QZNFRTVE**). Пример иллюстрирует работу с массивом структур, вложенным в другой массив структур.

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 OS/400, 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**
>
> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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

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
- Исходный код программы на Java, вызывающей QSYRUSRI

Исходный код PCML для вызова QSYRUSRI

```
<pcml version="1.0">

<!-- Исходный файл PCML для вызова API Получить информацию о пользователе (QSYRUSRI) -->

  <!-- Формат USRI0150 - Поддерживаются дополнительные форматы -->
  <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>

  <!-- Программа QSYRUSRI и список ее параметров для получения данных в формате USRI0100 -->
  <program name="qsyrusri" path="/QSYS.lib/QSYRUSRI.pgm">
    <data name="receiver"         type="struct"   struct="usri0100"   usage="output"/>
    <data name="receiverLength"   type="int"      length="4"          usage="input" />
```

```
    <data name="format"           type="char"     length="8"          usage="input"    init="USRI0100"/>
    <data name="profileName"      type="char"     length="10"         usage="input"    init="*CURRENT"/>
    <data name="errorCode"        type="int"      length="4"          usage="input"    init="0"/>
  </program>

</pcml>
```

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

Данный пример состоит из двух частей:

- Исходный код PCML для вызова QGYOLAUS
- Исходный код программы на Java, вызывающей QGYOLAUS

Исходный код PCML для вызова QGYOLAUS

```
<pcml version="1.0">

<!-- Исходный код PCML для вызова API Открыть список пользователей с правами доступа (QGYOLAUS) -->

  <!-- Формат AUTU0150 - Поддерживаются дополнительные форматы -->
  <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>

  <!-- Информационная структура списка (общая для всех API Открыть список...) -->
  <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>

  <!-- Программа QGYOLAUS и список ее параметров для получения данных в формате AUTU0150 -->
  <program name="qgyolaus" path="/QSYS.lib/QGY.lib/QGYOLAUS.pgm" parseorder="listInfo receiver">
    <data   name="receiver"       type="struct" struct="autu0150" usage="output"
              count="listInfo.rcdsReturned" outputsize="receiverLength" />
    <data   name="receiverLength" type="int"    length="4"        usage="input" init="16384" />
    <data   name="listInfo"       type="struct" struct="listInfo" usage="output" />
    <data   name="rcdsToReturn"   type="int"    length="4"        usage="input" init="264" />
    <data   name="format"         type="char"   length="10"       usage="input" init="AUTU0150" />
    <data   name="selection"      type="char"   length="10"       usage="input" init="*USER" />
    <data   name="member"         type="char"   length="10"       usage="input" init="*NONE" />
    <data   name="errorCode"      type="int"    length="4"        usage="input" init="0" />

  </program>

  <!-- 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>

  <!-- Программа QGYCLST закрывает список и освобождает ресурсы сервера -->
  <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

Данный пример состоит из двух частей:

- Исходный код PCML для вызова QZNFRTVE
- Исходный код программы на Java, вызывающей 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

В этом разделе перечислены примеры программ, встречающиеся в документации по классам создания отчетов IBM Toolbox for Java.

JSPReportProcessor и PDFContext

- Пример: Применение класса JSPReportProcessor с классом PDFContext
- Пример: Образец файла JSP класса JSPReportProcessor

XSLReportProcessor и PCLContext

- Пример: Применение класса XSLReportProcessor с PCLContext
- Пример: Файл XML для XSLReportProcessor
- Пример: Файл XSL для XSLReportProcessor

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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

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> <имя-файла-вывода>
//
////////////////////////////////////////////////////////////////////////////

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( Готов к анализу документа XSL );

        /** 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

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">Гостиница PAN PACIFIC
                                              HOTEL в Сан-Франциско </fo:block>
        <fo:block text-align="center"> Пятница, 8-9 декабря 2000 г. </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">Имя</fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="2">
               <fo:block border-bottom-style="solid">Адрес</fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="3">
               <fo:block border-bottom-style="solid">Город</fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="4">
               <fo:block border-bottom-style="solid">Штат</fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="5">
               <fo:block border-bottom-style="solid">ZIP-код</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

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

```
////////////////////////////////////////////////////////////////////////////
//
// В следующем примере (PCLRunReport) классы XSLPReportProcessor и
// PCLContext применяются для получения данных XML и их преобразования в формат PCL.
// Затем данные отправляются на принтер OutputQueue.
//
// Содержимое примеров исходных файлов XML и XSL, которые можно применять
// с PCLRunReport, можно просмотреть в realestate.xml и realestate.xsl.
// Вы можете также загрузить ZIP-файл с примерами файлов XML и XSL. Кроме того,
```

```
// ZIP-файл содержит пример файла JSP, который можно применять
// с примером класса JSPReportProcessor (JSPRunReport).
//
// Формат вызова:
//     java PCLRunReport <имя-файла-xml> <имя-файла-xsl>
//
////////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.awt.*;
import java.io.*;
import java.awt.print.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.HashMap;

import com.ibm.xsl.composer.flo.*;
import com.ibm.xsl.composer.areas.*;
import com.ibm.xsl.composer.framework.*;
import com.ibm.xsl.composer.java2d.*;
import com.ibm.xsl.composer.prim.*;
import com.ibm.xsl.composer.properties.*;
import com.ibm.as400.util.reportwriter.processor.*;
import com.ibm.as400.util.reportwriter.pclwriter.*;
import java.io.IOException;
import java.io.Serializable;
import org.xml.sax.SAXException;
import com.ibm.as400.access.*;

public class PCLRunReport

{

    public static void main( String args[] )
    {
        SpooledFileOutputStream fileout = null;
        String xmldocumentName = args[0];
        String xsldocumentName = args[1];

        String sys = "<system>";      /* Укажите имя сервера ISeries                    */
        String user = "<user>";       /* Укажите имя пользовательского профайла ISeries */
        String pass = "<password>";   /* Укажите пароль ISeries                         */

        AS400 system = new AS400(sys, user, pass);

        /* Укажите очередь вывода iSeries */
        String outqname = "/QSYS.LIB/qusrsys.LIB/<outq>.OUTQ";
        OutputQueue outq = new OutputQueue(system, outqname);
        PrintParameterList parms = new PrintParameterList();
        parms.setParameter(PrintObject.ATTR_OUTPUT_QUEUE, outq.getPath());

        try{
          fileout = new SpooledFileOutputStream(system, parms, null, null);
        }
        catch (Exception e)
        {}

        /** настройка формата страницы **/
        Paper paper = new Paper();
        paper.setSize(612,792);
        paper.setImageableArea(18, 36, 576, 720);
        PageFormat pf = new PageFormat();
        pf.setPaper(paper);

        /** создание объекта PCLContext и указание FileOutputStream
            в качестве объекта OutputStream **/
```

```
        PCLContext pclcontext = new PCLContext((OutputStream)fileout, pf);

        System.out.println("Программа готова к анализу документа XSL");

        /** создание объекта XSLReportProcessor **/
        XSLReportProcessor xslprocessor = new XSLReportProcessor(pclcontext);
        try {
        xslprocessor.setXMLDataSource(xmldocumentName);
        }
        catch (SAXException se) {
            String mes = se.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        catch (IOException ioe) {
            String mes = ioe.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        catch (NullPointerException np){
            String mes = np.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        /** настройка шаблона согласно указанному источнику данных XML **/
        try {
        xslprocessor.setTemplate(xsldocumentName);
        }
        catch (NullPointerException np){
            String mes = np.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        catch (IOException e) {
            String mes = e.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        catch (SAXException se) {
            String mes = se.getMessage();
            System.out.println(mes);
            System.exit(0);
            }

        /** обработка отчета **/
        try {
        xslprocessor.processReport();
        }
        catch (IOException e) {
            String mes = e.getMessage();
            System.out.println(mes);
            System.exit(0);
            }
        catch (SAXException se) {
            String mes = se.getMessage();
            System.out.println(mes);
            System.exit(0);
            }

        System.exit(0);
    }

}
```

Пример: Пример файла XML для класса XSLReportProcessor

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>Квартира</TYPE>
 <PRICE>$110000</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>Некоторая ул., 13</ADDRESS>
    <CITY>Дорчестер</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>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>
 <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

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

В этом разделе перечислены примеры программ, встречающиеся в документации по классам ресурсов IBM Toolbox for Java.

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

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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

Ниже приведены примеры различных приемов работы со списками ресурсов:

- 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

Одним из действительных классов, дочерних по отношению к классу ResourceList является com.ibm.as400.resource.RJobList, соответствующий списку заданий iSeries. Класс RJobList поддерживает множество ИД выбора и ИД сортировки, предназначенных для фильтрации и сортировки списка. В данном примере содержимое объекта RJobList выводится на принтер:

```
// 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:

> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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: 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

Действительный класс com.ibm.as400.resource.RUser, соответствующий пользователю iSeries, является дочерним по отношению к классу Resource. Класс RUser поддерживает множество ИД атрибутов, обеспечивающих доступ к значениями атрибутов.

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);
```

Пример: Изменение значений атрибутов ресурса ChangeableResource

Действительный класс com.ibm.as400.resource.RJob, соответствующий заданию iSeries, является дочерним по отношению к классу ChangeableResource. Класс RJob поддерживает множество ИД атрибутов, обеспечивающих доступ к значениями атрибутов. 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();
```

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.

Каждому атрибуту соответствует объект мета-данных атрибута (com.ibm.as400.resource.ResourceMetaData), который описывает различные свойства атрибута. Эти свойства включают возможные значения атрибута и значения по умолчанию, а также возможность изменения значения атрибута.

Примеры:

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();
}
```

Примеры: RFML

В этом разделе перечислены примеры программ, встречающиеся в документации по компоненту IBM ToolboxME for Java RFML:

- Пример: Сравнение RFML с классами IBM Toolbox for Java Record
- Пример: Исходный файл RFML

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

> **Отказ от гарантий на предоставляемый код**
>
> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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

В этом примере исходного файла RFML определяется формат записей заказчиков, используемых в примере RFML Сравнение RFML с классами Record IBM Toolbox for Java. 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>
```

Пример: изменение владельца нити OS/400 с помощью разрешения

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

Ниже приведен фрагмент кода, в котором профайлу пользователя выдается одноразовое разрешение, позволяющее заменить владельца нити OS/400 и работать в системе от имени этого пользователя:

```
// Подготовка к работе с локальной системой AS/400
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 OS/400 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 OS/400
cr.swap();

// Удаление разрешений
cr.destroy();
pt.destroy();
```

Примеры работы с классами сервлетов

Ниже приведены примеры работы с классами сервлетов:

- 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:

> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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

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” на стр. 641
- “Представление кода HTML в браузере” на стр. 641

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. 240 в примере на Java, приведенном выше, создается следующий код HTML.

```
<table>
<tr>
<th>ИД заказчика</th>
<th>Номер заказа</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

Пример состоит из трех частей:

- Исходный код на Java, иллюстрирующий применение класса RecordListRowData
- Исходный код HTML, созданный кодом на Java с помощью класса HTMLTableConverter
- Представление кода HTML в браузере

Исходный код на Java, демонстрирующий работу класса RecordListRowData

```
// 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

Данный пример состоит из трех частей:

- Исходного кода на Java, иллюстрирующего работу класса SQLResultSetRowData
- Исходный код HTML, созданный кодом на Java с помощью класса HTMLTableConverter
- Представление кода HTML в браузере

Исходный код на Java, демонстрирующий работу класса SQLRecordListRowData

```
// 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
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

Откомпилируйте и запустите этот пример, иллюстрирующий применение класса HTMLFormConverter, когда в системе активен Web-сервер с поддержкой сервлетов.

```
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 = "Пример применения HTMLFormConverter";
   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>В этом примере используется класс HTMLFormConverter " +
                      "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 " +
                      "записей.</P>");
      mainForm.addElement(desc);

      // Добавление инструкций в форму.
      HTMLText instr =
         new HTMLText("<P>Введите имя сервера и " +
                      "имена файла и библиотеки, которые нужно " +
                      "считать.  После этого нажмите кнопку Показать записи " +
                      "для продолжения.</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>Номер записи: 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><!-- Вывод класса HTMLFormConverter-->
</b></font></td>
</tr>
</table>
</td>
</tr>
<tr>
<form>
<td><input type="submit" name="getFirstRecord" value="Первая" />
<input type="submit" name="getPreviousRecord" value="Предыдущая" />
<input type="submit" name="getNextRecord" value="Следующая" />
<input type="submit" name="getLastRecord" value="Последняя" />
<br />
</td>
</tr>
<tr>
<td><input type="submit" name="returnToMain" value="Возврат в основную форму" />
<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 и 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>Не зарегистрировано ни одного автомобиля.</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>Не зарегистрировано ни одного автомобиля.</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 для Java. Эти примеры снабжены подробными пояснениями, что позволяет их использовать даже при отсутствии опыта работы с классами Toolbox for Java.

Если вы только начинаете работу с продуктом и вам требуется помощь, обратитесь к разделу Создание первой программы с помощью IBM Toolbox for Java.

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

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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 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

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 класса CommandCall IBM Toolbox for Java
//
// Пример списка заданий IBM Toolbox for Java.
//
///////////////////////////////////////////////////////////////////////////
//
// Классы доступа IBM Toolbox for Java содержатся в пакете
// 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 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 OS/400. 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)

[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.
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. IBM Toolbox for Java создает нити приложений, с помощью которых выполняются операции IBM Toolbox for Java. 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)

[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, пользователю будет предложено ввести свой идентификатор и пароль. **Примечание:** если программа будет вызвана без имени системы AS/400, то объект AS400 запросит его у пользователя.
   - Если программа будет запущена на JVM системы iSeries, то будут применяться ИД и пароль пользователя, запустившего программу на Java. 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)

[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. Для этого применяется объект **MessageQueue** IBM Toolbox для Java. 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)

[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)

[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 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 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)

[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 JDBCPopulate.  Данная программа создает и заполняет таблицу
// 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)

[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>"; Примечание 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

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 класса Vaccess IBM Toolbox for Java
//
//
// Пример списка заданий IBM Toolbox for Java.
//
////////////////////////////////////////////////////////////////////////////

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.
2. Эта строка делает доступными все классы графического интерфейса 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 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 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

В этом разделе перечислены примеры программ, встречающиеся в советах по программированию IBM Toolbox for Java.

Управление соединениями

- Пример: Создание соединения с iSeries с помощью объекта CommandCall
- Пример: Настройка двух соединений с iSeries с помощью объекта CommandCall
- Пример: Создание объектов CommandCall и IFSFileInputStream objects с помощью объекта AS400
- Пример: Предварительное подключение к серверу iSeries с помощью класса AS400ConnectionPool
- Пример: Предварительное подключение к службе сервера iSeries с помощью класса AS400ConnectionPool и повторное использование соединения

Подключение и отключение

- Пример: Предварительное подключение программы на Java к серверу iSeries
- Пример: Отключение программы на Java от сервера iSeries
- Пример: Отключение программы на Java от сервера iSeries и ее повторное подключение с помощью функций disconnectService() и run()
- Пример: Отключение программы на Java от сервера iSeries без возможности повторного подключения

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

Установка и обновление

- Пример: Использование класса AS400ToolboxInstaller

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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: ToolboxME for iSeries

В этом разделе перечислены примеры программ, встречающиеся в документации по IBM ToolboxME for iSeries.

- “Пример программы ToolboxME for iSeries: JdbcDemo.java” на стр. 363
- “Пример: Применение ToolboxME for iSeries, MIDP и JDBC”
- “Пример: Применение ToolboxME for iSeries, MIDP и IBM Toolbox for Java” на стр. 694

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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 ToolboxME for iSeries, MIDP и JDBC

Следующий пример исходного кода демонстрирует, каким образом приложение ToolboxME for iSeries может с помощью Профайла мобильных устройств (MIDP) и JDBC получить доступ к базе данных и сохранить информацию в автономной памяти.

Программа, рассмотренная в данном примере, предназначена для агента по продаже недвижимости, которому необходимо просматривать и делать заявки на объекты, выставленные на продажу. Агент получает доступ к информации, хранящейся в базе данных сервера iSeries, с устройства Tier0.

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.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример программы ToolboxME for iSeries. Это программа MIDlet, демонстрирующая,
// 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 = "выберите mls, адрес, текущую заявку из
                      qjdbcme.realestate, где текущая заявка <> 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)
    {
    }
}
```

Пример: Применение ToolboxME for iSeries, MIDP и IBM Toolbox for Java

Следующий пример исходного кода демонстрирует, каким образом приложение ToolboxME for iSeries может с помощью Профайла мобильных устройств (MIDP) и IBM Toolbox for Java получить доступ к данным и службам сервера iSeries.

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.

Рабочая программа, получаемая в результате преобразования исходного кода, запускает команды на сервере iSeries с помощью файла Языка описаний вызовов программ (PCML).

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.

```
////////////////////////////////////////////////////////////////////////////
//
// Пример программы ToolboxME for iSeries. Это программа демонстрирует,
// каким образом ToolboxME for iSeries может с помощью файла PCML получать
// доступ к данным и службам на сервере iSeries.
//
// Для работы приложения необходимо, чтобы файл qsyrusri.pcml
// был указан в разделе CLASSPATH сервера MEServer.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.sql.*;
import java.util.Hashtable;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.*;

import com.ibm.as400.micro.*;


public class ToolboxMidpDemo extends MIDlet implements CommandListener
{
    private Display   display_;

    // Системный объект ToolboxME.
    private AS400 system_;

    private List      main_ = new List("Демонстрационная версия ToolboxME MIDP", Choice.IMPLICIT);

    // Создание формы для каждого компонента.
    private Form      signonForm_;
    private Form      cmdcallForm_;
```

```
private Form      pgmcallForm_;
private Form      dataqueueForm_;
private Form      aboutForm_;

// Видимый текст для каждого компонента.
static final String SIGN_ON       = "Вход в систему";
static final String COMMAND_CALL  = "Вызов команды";
static final String PROGRAM_CALL  = "Вызов программы";
static final String DATA_QUEUE    = "Очередь данных";
static final String ABOUT         = "О программе";

static final String NOT_SIGNED_ON = "Не зарегистрирован в системе.";
static final String DQ_READ       = "Чтение";
static final String DQ_WRITE      = "Запись";

// Индикатор состояния входа в систему.
private Ticker    ticker_ = new Ticker(NOT_SIGNED_ON);

// Команды, которые можно выполнять.
private static final Command actionExit_   = new Command("Выход", Command.SCREEN, 0);
private static final Command actionBack_   = new Command("Назад", Command.SCREEN, 0);
private static final Command actionGo_     = new Command("Далее", Command.SCREEN, 1);
private static final Command actionClear_  = new Command("Очистить", Command.SCREEN, 1);
private static final Command actionRun_    = new Command("Выполнить", Command.SCREEN, 1);
private static final Command actionSignon_ = new Command(SIGN_ON, Command.SCREEN, 1);
private static final Command actionSignoff_= new Command("Выход из системы", Command.SCREEN, 1);

private Displayable   onErrorGoBackTo_;  // форма, возвращаемая по окончании
                                         // просмотра формы ошибок

// Поля TextField для формы SignOn.
private TextField signonSystemText_ = new TextField("Система", "rchasdm3", 20,
                                                    TextField.ANY);
private TextField signonUidText_ = new TextField("ИД пользователя", "JAVA", 10,
                                                 TextField.ANY);
// Временный TBD
private TextField signonPwdText_ = new TextField("Пароль", "JTEAM1", 10,
                                                 TextField.PASSWORD);
private TextField signonServerText_ = new TextField("Сервер ME Server", "localhost",
                                                    10, TextField.ANY);
private StringItem signonStatusText_ = new StringItem("Состояние", NOT_SIGNED_ON);

// Поля TextField для формы CommandCall.
// TBD: максимальный размер; TBD: Текстовое описание???
private TextField cmdText_ = new TextField("Команда", "CRTLIB FRED", 256,
                                           TextField.ANY);
private StringItem cmdMsgText_ = new StringItem("Сообщения", null);
private StringItem cmdStatusText_ = new StringItem("Состояние", null);

// Поля TextField для формы ProgramCall.
private StringItem pgmMsgDescription_ = new StringItem("Сообщения", null);
private StringItem pgmMsgText_ = new StringItem("Сообщения", null);

// Поля TextField для формы DataQueue.
private TextField dqInputText_ = new TextField("Данные для записи",
                                   "Здравствуйте", 30, TextField.ANY);
private StringItem dqOutputText_ = new StringItem("Содержимое очереди данных",
                                                  null);
private ChoiceGroup dqReadOrWrite_ = new ChoiceGroup("Действие",
                                   Choice.EXCLUSIVE,
                                   new String[] { DQ_WRITE, DQ_READ},
                                   null);
private StringItem dqStatusText_ = new StringItem("Состояние", null);


/**
 * Создание нового ToolboxMidpDemo.
```

```
 **/
public ToolboxMidpDemo()
{
    display_ = Display.getDisplay(this);
    // Примечание: В примере с KVM главная панель была создана с помощью TabbedPane.
    // MIDP не содержит похожих классов, поэтому будет использован List.
}

/**
 * Показ главного меню.
 * Реализация абстрактного метода класса Midlet.
 **/
protected void startApp()
{
    main_.append(SIGN_ON, null);
    main_.append(COMMAND_CALL, null);
    main_.append(PROGRAM_CALL, null);
    main_.append(DATA_QUEUE, null);
    main_.append(ABOUT, null);

    main_.addCommand(actionExit_);
    main_.setCommandListener(this);

    display_.setCurrent(main_);
}

// Реализация метода интерфейса CommandListener.
public void commandAction(Command action, Displayable dsp)
{
    // Обработка всех действий 'exit' и 'back' одинакова.
    if (action == actionExit_)
    {
        destroyApp(false);

        notifyDestroyed();
    }
    else if (action == actionBack_)
    {
        // Возврат в главное меню.
        display_.setCurrent(main_);
    }
    else if (dsp instanceof List)
    {
        List current = (List)dsp;

        // Действие произошло на главной странице
        if (current == main_)
        {
            int   idx = current.getSelectedIndex();

            switch (idx)
            {
            case 0:     // SignOn
                showSignonForm();
                break;
            case 1:     // CommandCall
                showCmdForm();
                break;
            case 2:     // ProgramCall
                showPgmForm();
                break;
            case 3:     // DataQueue
                showDqForm();
                break;
            case 4:     // About
                showAboutForm();
                break;
```

```
        default:    // Ни одно из вышеперечисленных
            feedback("Внутренняя ошибка: необработанные элементы индекса в main: " + idx,
                     AlertType.ERROR);
            break;
        }
    } // current == main
    else
        feedback("Внутренняя ошибка: просматриваемый объект имеет тип
                  List, но не относится к main_.",
                 AlertType.ERROR);
} // instanceof List
else if (dsp instanceof Form)
{
    Form current = (Form)dsp;

    if (current == signonForm_)
    {
        if (action == actionSignon_)
        {
            // Создание системного объекта ToolboxME.
            system_ = new AS400(signonSystemText_.getString(),
                                signonUidText_.getString(),
                                signonPwdText_.getString(),
                                signonServerText_.getString());

            try
            {
                // Подключение к iSeries.
                system_.connect();

                // Настройка текста о состоянии входа в систему.
                signonStatusText_.setText("Вход в систему выполнен.");

                // Показать окно подтверждения входа в систему.
                feedback("Регистрация в системе успешно выполнена.", AlertType.INFO, main_);

                // Заменять кнопку SignOn на кнопку SignOff.
                signonForm_.removeCommand(actionSignon_);
                signonForm_.addCommand(actionSignoff_);

                // Обновить индикатор.
                ticker_.setString("... Вошел в систему '" +
                                  signonSystemText_.getString() + "' как '" +
                                  signonUidText_.getString() + "' с помощью '" +
                                  signonServerText_.getString() + "' ... ");
            }
            catch (Exception e)
            {
                e.printStackTrace();

                // Настройка текста о состоянии входа в систему.
                signonStatusText_.setText(NOT_SIGNED_ON);

                feedback("Регистрация в системе не выполнена.  " +
                         e.getMessage(), AlertType.ERROR);
            }
        }
        else if (action == actionSignoff_)
        {
            if (system_ == null)
                feedback("Внутренняя ошибка: пустая система.", AlertType.ERROR);
            else
            {
                try
                {
                    // Отключение от iSeries.
                    system_.disconnect();
```

```
                    system_ = null;

                    // Настройка текста о состоянии входа в систему.
                    signonStatusText_.setText(NOT_SIGNED_ON);

                    // Показать окно подтверждения выхода из системы.
                    feedback("Сеанс работы успешно завершен.", AlertType.INFO, main_);

                    // Заменить кнопку SignOff на кнопку SignOn.
                    signonForm_.removeCommand(actionSignoff_);
                    signonForm_.addCommand(actionSignon_);

                    // Обновить индикатор.
                    ticker_.setString(NOT_SIGNED_ON);
                }
                catch (Exception e)
                {
                    feedback(e.toString(), AlertType.ERROR);

                    e.printStackTrace();

                    signonStatusText_.setText("Ошибка.");

                    feedback("Ошибка при завершении сеанса работы.", AlertType.ERROR);
                }
            }
        }
        else  // Ни одно из вышеперечисленных.
        {
            feedback("Внутренняя ошибка: неизвестное действие.", AlertType.INFO);
        }
    } // signonForm_
    else if (current == cmdcallForm_)
    {
        if (action == actionRun_)
        {
            // Если пользователь не вошел в систему, выдать предупреждение.
            if (system_ == null)
            {
                feedback(NOT_SIGNED_ON, AlertType.ERROR);
                return;
            }

            // Получить команду, введенную пользователем на беспроводном устройстве.
            String cmdString = cmdText_.getString();

            // Если команда не задана, выдать предупреждение.
            if (cmdString == null || cmdString.length() == 0)
                feedback("Укажите команду.", AlertType.ERROR);
            else
            {
                try
                {
                    // Запуск команды.
                    String[] messages = CommandCall.run(system_, cmdString);

                    StringBuffer status = new StringBuffer("Команда выполнена со значениями ");

                    // Проверить, нет ли сообщений
                    if (messages.length == 0)
                    {
                        status.append("ответных сообщений нет.");

                        cmdMsgText_.setText(null);

                        cmdStatusText_.setText("Команда успешно выполнена.");
                    }
```

```
                else
                {
                    if (messages.length == 1)
                        status.append("Получено одно сообщение.");
                    else
                        status.append(messages.length + " сообщений получено.");

                    // Если есть сообщения, показать только первое.
                    cmdMsgText_.setText(messages[0]);

                    cmdStatusText_.setText(status.toString());
                }

                repaint();
            }
            catch (Exception e)
            {
                feedback(e.toString(), AlertType.ERROR);

                e.printStackTrace();

                feedback("Ошибка при выполнении команды.", AlertType.ERROR);
            }
        }
    }
    else if (action == actionClear_)
    {
        // Стереть текст команды и сообщения.
        cmdText_.setString("");

        cmdMsgText_.setText(null);

        cmdStatusText_.setText(null);

        repaint();
    }
    else  // Ни одно из вышеперечисленных.
    {
        feedback("Внутренняя ошибка: Неизвестное действие.", AlertType.INFO);
    }
} // cmdcallForm_
else if (current == pgmcallForm_)
{
    if (action == actionRun_)
    {
        // Если пользователь не вошел в систему перед вызовом программы, выдать предупреждение.
        if (system_ == null)
        {
            feedback(NOT_SIGNED_ON, AlertType.ERROR);
            return;
        }

        pgmMsgText_.setText(null);

        // Обратитесь к примеру PCML в документации к IBM Toolbox for Java.
        String pcmlName = "qsyrusri.pcml"; // Файл PCML, который следует использовать.
        String apiName = "qsyrusri";

        // Создать хэш-таблицу с входными параметрами для вызова программы.
        Hashtable parmsToSet = new Hashtable(2);
        parmsToSet.put("qsyrusri.receiverLength", "2048");
        parmsToSet.put("qsyrusri.profileName", signonUidText_.getString().toUpperCase());

        // Создать строковый массив для получаемых выходных параметров.
        String[] parmsToGet = { "qsyrusri.receiver.userProfile",
                                "qsyrusri.receiver.previousSignonDate",
                                "qsyrusri.receiver.previousSignonTime",
```

```
                              "qsyrusri.receiver.daysUntilPasswordExpires"};

            // Строковый массив с описаниями выдаваемых параметров.
            String[] displayParm = { "Профайл",
                                     "Дата последнего входа в систему",
                                     "Время последнего входа в систему",
                                     "Срок действия пароля истек (дней)"};

            try
            {
                // Запуск программы.
                String[] valuesToGet = ProgramCall.run(system_,
                                                       pcmlName,
                                                       apiName,
                                                       parmsToSet,
                                                       parmsToGet);

                // Создать StringBuffer и занести в него все полученные параметры.
                StringBuffer txt = new StringBuffer();
                txt.append(displayParm[0] + ": " + valuesToGet[0] + "\n");

                char[] c = valuesToGet[1].toCharArray();
                txt.append(displayParm[1] + ": " + c[3] + c[4] + "/" +
                           c[5] + c[6] + "/" + c[1] + c[2] + "\n");

                char[] d = valuesToGet[2].toCharArray();
                txt.append(displayParm[2] + ": " + d[0] + d[1] + ":" + d[2] + d[3] + "\n");
                txt.append(displayParm[3] + ": " + valuesToGet[3] + "\n");

                // Задать отображаемый текст для результатов вызова программы.
                pgmMsgText_.setText(txt.toString());

                StringBuffer status = new StringBuffer("Программа выполнена со значениями ");

                if (valuesToGet.length == 0)
                {
                    status.append("значения не получены.");

                    feedback(status.toString(), AlertType.INFO);
                }
                else
                {
                    if (valuesToGet.length == 1)
                        status.append("Получено одно значение.");
                    else
                        status.append(valuesToGet.length + " значений получено.");

                    feedback(status.toString(), AlertType.INFO);
                }
            }
            catch (Exception e)
            {
                feedback(e.toString(), AlertType.ERROR);

                e.printStackTrace();

                feedback("Ошибка при выполнении программы.", AlertType.ERROR);
            }
        }
        else if (action == actionClear_)
        {
            // Очистка результатов вызова программы.
            pgmMsgText_.setText(null);

            repaint();
        }
    }  // pgmcallForm_
```

```
else if (current == dataqueueForm_)  // DataQueue
{
    if (action == actionGo_)
    {
        // Если перед выполнением операций с очередями данных пользователь не вошел в систему,
        // показать предупреждающее сообщение.
        if (system_ == null)
        {
            feedback(NOT_SIGNED_ON, AlertType.ERROR);

            return;
        }

        // Создать библиотеку для очереди данных.
        try
        {
            CommandCall.run(system_, "CRTLIB FRED");
        }
        catch (Exception e)
        {
        }

        // Выполнить команду создания очереди данных.
        try
        {
            CommandCall.run(system_, "CRTDTAQ FRED/MYDTAQ MAXLEN(2000)");
        }
        catch (Exception e)
        {
            feedback("Ошибка при создании очереди данных.  " + e.getMessage(),
                     AlertType.WARNING);
        }

        try
        {
            // Определение выбранного действия (Чтение или Запись).
            if (dqReadOrWrite_.getString(dqReadOrWrite_.getSelectedIndex()).equals(DQ_WRITE))
            {
                // Запись
                dqOutputText_.setText(null);

                // Внести текстовые входные данные беспроводного устройства в
                // очередь сообщений.
                if (dqInputText_.getString().length() == 0)
                    dqStatusText_.setText("Данные не заданы.");
                else
                {
                    // Запись в очередь данных.
                    DataQueue.write(system_,
                                    "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ",
                                    dqInputText_.getString().getBytes() );

                    dqInputText_.setString(null);

                    // Показать состояние.
                    dqStatusText_.setText("Запись выполнена успешно.");
                }
            }
            else  // Чтение
            {
                // Чтение из очереди данных.
                byte[] b = DataQueue.readBytes(system_, "/QSYS.LIB/FRED.LIB/MYDTAQ.DTAQ");

                // Определить, содержала ли очередь сообщений данные, и
                // вывести соответствующее сообщение.
                if (b == null)
                {
```

```
                            dqStatusText_.setText("Нет доступных записей очереди данных.");

                            dqOutputText_.setText(null);
                        }
                        else if (b.length == 0)
                        {
                            dqStatusText_.setText("Запись очереди данных пуста.");

                            dqOutputText_.setText(null);
                        }
                        else
                        {
                            dqStatusText_.setText("Чтение выполнено успешно.");

                            dqOutputText_.setText(new String(b));
                        }
                    }

                    repaint();
                }
                catch (Exception e)
                {
                    e.printStackTrace();

                    feedback(e.toString(), AlertType.ERROR);

                    feedback("Ошибка при выполнении команды.  " + e.getMessage(), AlertType.ERROR);
                }
            }  // actionGo_
            else if (action == actionClear_)
            {
                // Очистка формы очереди данных.
                dqInputText_.setString("");

                dqOutputText_.setText(null);

                dqReadOrWrite_.setSelectedFlags(new boolean[] { true, false});

                dqStatusText_.setText(null);

                repaint();
            }
            else  // Ни одно из вышеперечисленных.
            {
                feedback("Внутренняя ошибка: неизвестное действие.", AlertType.INFO);
            }
        } // dataqueueForm_
        else if (current == aboutForm_)  // "О программе".
        {
            // Для выполнения этой части кода необходимо воспользоваться кнопкой "Назад".
        }  // Ни одно из вышеперечисленных.
        else
            feedback("Внутренняя ошибка: неизвестная форма.", AlertType.ERROR);
    } // instanceof Form
    else
        feedback("Внутренняя ошибка: неизвестный просматриваемый объект.", AlertType.ERROR);
}



/**
 *  Вывод формы "О программе".
 **/
private void showAboutForm()
{
    // Если форма О программе пуста, создать и добавить ее.
    if (aboutForm_ == null)
```

```
    {
        aboutForm_ = new Form(ABOUT);
        aboutForm_.append(new StringItem(null,
                          "Это пример приложения MIDP, в котором применяется " +
                          "IBM Toolbox for Java Micro Edition (ToolboxME)."));

        aboutForm_.addCommand(actionBack_);
        aboutForm_.setCommandListener(this);
    }

    display_.setCurrent(aboutForm_);
}



/**
 *  Вывод формы "Вход в систему".
 **/
private void showSignonForm()
{
    // Создание формы входа в систему.
    if (signonForm_ == null)
    {
        signonForm_ = new Form(SIGN_ON);
        signonForm_.append(signonSystemText_);
        signonForm_.append(signonUidText_);
        signonForm_.append(signonPwdText_);
        signonForm_.append(signonServerText_);
        signonForm_.append(signonStatusText_);
        signonForm_.addCommand(actionBack_);
        signonForm_.addCommand(actionSignon_);
        signonForm_.setCommandListener(this);
        signonForm_.setTicker(ticker_);
    }

    display_.setCurrent(signonForm_);
}



/**
 *  Вывод формы "Вызов команды".
 **/
private void showCmdForm()
{
    // Создание формы вызова команды.
    if (cmdcallForm_ == null)
    {
        cmdcallForm_ = new Form(COMMAND_CALL);
        cmdcallForm_.append(cmdText_);
        cmdcallForm_.append(cmdMsgText_);
        cmdcallForm_.append(cmdStatusText_);
        cmdcallForm_.addCommand(actionBack_);
        cmdcallForm_.addCommand(actionClear_);
        cmdcallForm_.addCommand(actionRun_);
        cmdcallForm_.setCommandListener(this);
        cmdcallForm_.setTicker(ticker_);
    }

    display_.setCurrent(cmdcallForm_);
}



/**
 *  Вывод формы "Вызов программы".
 **/
```

```
private void showPgmForm()
{
    // Создание формы вызова программы.
    if (pgmcallForm_ == null)
    {
        pgmcallForm_ = new Form(PROGRAM_CALL);
        pgmcallForm_.append(new StringItem(null,
                 "Будет вызван API Получить информацию о пользователе (QSYRUSRI) " +
                 "API, после чего будут получены сведения о текущем " +
                 "пользовательском профайле."));
        pgmcallForm_.append(pgmMsgText_);
        pgmcallForm_.addCommand(actionBack_);
        pgmcallForm_.addCommand(actionClear_);
        pgmcallForm_.addCommand(actionRun_);
        pgmcallForm_.setCommandListener(this);
        pgmcallForm_.setTicker(ticker_);
    }

    display_.setCurrent(pgmcallForm_);
}


/**
 *  Вывод формы "Очередь данных".
 **/
private void showDqForm()
{
    // Создание формы очереди данных.
    if (dataqueueForm_ == null)
    {
        dataqueueForm_ = new Form(DATA_QUEUE);
        dataqueueForm_.append(dqInputText_);
        dataqueueForm_.append(dqOutputText_);
        dataqueueForm_.append(dqReadOrWrite_);
        dataqueueForm_.append(dqStatusText_);
        dataqueueForm_.addCommand(actionBack_);
        dataqueueForm_.addCommand(actionClear_);
        dataqueueForm_.addCommand(actionGo_);
        dataqueueForm_.setCommandListener(this);
        dataqueueForm_.setTicker(ticker_);
    }

    display_.setCurrent(dataqueueForm_);
}


private void feedback(String text, AlertType type)
{
    feedback(text, type, display_.getCurrent());
}

/**
 *  Этот метод применяется для создания окна диалога и вывода
 *  уведомления с помощью отправки пользователю предупреждающего сообщения.
 **/
private void feedback(String text, AlertType type, Displayable returnToForm)
{
    System.err.flush();
    System.out.flush();

    Alert alert = new Alert("Alert", text, null, type);

    if (type == AlertType.INFO)
        alert.setTimeout(3000);  // время в миллисекундах
    else
        alert.setTimeout(Alert.FOREVER);  // Предупреждение аннулируется только пользователем.
```

```
        display_.setCurrent(alert, returnToForm);
    }


    // Принудительная перерисовка текущей формы.
    private void repaint()
    {
        Alert alert = new Alert("Обновление меню...", null, null, AlertType.INFO);
        alert.setTimeout(1000);  // время в миллисекундах

        display_.setCurrent(alert, display_.getCurrent());
    }



    /**
     * Пауза, освобождение ненужного сейчас пространства.
     * Реализация абстрактного метода класса Midlet.
     **/
    protected void pauseApp()
    {
        display_.setCurrent(null);
    }



    /**
     * Общая очистка.
     * Реализация абстрактного метода класса Midlet.
     **/
    protected void destroyApp(boolean unconditional)
    {
        // Отключение от iSeries, если выполняется уничтожение или завершение работы с Midlet
        if (system_ != null)
        {
            try
            {
                system_.disconnect();
            }
            catch (Exception e)
            {
            }
        }
    }
}
```

Примеры: Классы Utility

В этом разделе перечислены примеры программ, встречающиеся в документации по классам утилит IBM Toolbox for Java

Класс AS/400ToolboxInstaller

- Пример: Использование класса AS400ToolboxInstaller
- Пример: Установка IBM Toolbox for Java с помощью AS400ToolboxInstaller
- Пример: Установка пакета ACCESS из командной строки
- Пример: Работа с классом Graphical Toolbox из командной строки

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**
>
> Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.
>
> Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.
>
> 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 AS400ToolboxInstaller для установки и обновления IBM Toolbox for Java

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

```
////////////////////////////////////////////////////////////////////////////
//
// Пример установки и обновления IBM Toolbox for Java. Эта программа применяет
// класс AS400ToolboxInstaller установки и обновления на рабочей станции
// пакета IBM Toolbox for Java.
//
// Программа проверяет наличие пакета IBM Toolbox for Java в целевом каталоге.
// Если пакет не будет найден, программа установит пакет на рабочей станции.
// Если пакет будет найден, и в исходном каталоге есть обновления, программа
// скопирует эти обновления на рабочую станцию.
//
// Формат вызова:
//    checkToolbox исходный-каталог целевой-каталог
//
// Где
// исходный-каталог = имя каталога с исходными файлами в формате URL.
// целевой-каталог  = имя каталога с целевыми файлами.
//
////////////////////////////////////////////////////////////////////////////

import java.io.*;
import java.util.*;
import java.net.*;
import utilities.*;

public class checkToolbox extends Object
{
   public static void main(String[] parameters)
   {
      System.out.println( " " );



      // Продолжить процедуру установки/обновления, только
      // если заданы оба каталога.
```

```
if (parameters.length >= 2)
{

   // Первый параметр - исходный каталог, второй - целевой каталог.

   String sourcePath = parameters[0];
   String targetPath = parameters[1];

   boolean installIt = false;
   boolean updateIt  = false;



   // Создание программы чтения ввода пользователя.

   BufferedReader inputStream = new BufferedReader(new InputStreamReader(System.in),1);

   try
   {
      // Ссылка на исходный пакет. AS400ToolboxInstaller
      // применяет для доступа к файлам класс URL.

      URL sourceURL  = new URL(sourcePath);



      // Проверка наличия пакета на клиентском компьютере. Если пакет
      // не установлен - вывод запроса на установку пакета.

      if (AS400ToolboxInstaller.isInstalled("ACCESS", targetPath) == false)
      {
         System.out.print("Продукт IBM Toolbox for Java не установлен. Установить сейчас? (Y/N):");

         String userInput = inputStream.readLine();

         if ((userInput.charAt(0) == 'y')   ||
             (userInput.charAt(0) == 'Y'))
            installIt = true;
      }



      // Пакет установлен. Проверка наличия на сервере новых обновлений.
      // Если такие обновления есть - вывод запроса на их установку.

      else
      {
         if (AS400ToolboxInstaller.isUpdateNeeded("ACCESS", targetPath, sourceURL) == true)
         {
            System.out.print("Для продукта IBM Toolbox for Java существуют исправления.
            Установить сейчас? (Y/N):");

            String userInput = inputStream.readLine();

            if ((userInput.charAt(0) == 'y') || (userInput.charAt(0) == 'Y'))
               updateIt = true;
         }
         else
            System.out.println("Целевой каталог совпадает с текущим, обновление не требуется.");
      }



      // Если пакет нужно установить или обновить.

      if (updateIt || installIt)
      {
```

```
            // Скопировать файлы с сервера в целевой каталог.

            AS400ToolboxInstaller.install("ACCESS", targetPath, sourceURL);



            // Сообщить о завершении установки или обновления.

            System.out.println(" ");

            if (installIt)
               System.out.println("Установка выполнена!");
            else
               System.out.println("Обновление выполнено!");



            // Вывод значений, которые следует добавить в переменную среды CLASSPATH.

            Vector classpathAdditions = AS400ToolboxInstaller.getClasspathAdditions();

            if (classpathAdditions.size() > 0)
            {
               System.out.println("");
               System.out.println("Добавьте следующие объекты в переменную среды CLASSPATH:");

               for (int i = 0; i < classpathAdditions.size(); i++)
               {
                  System.out.print("   ");
                  System.out.println((String)classpathAdditions.elementAt(i));
               }
            }



            // Вывод значений, которые следует удалить из переменной среды CLASSPATH.

            Vector classpathRemovals = AS400ToolboxInstaller.getClasspathRemovals();

            if (classpathRemovals.size() > 0)
            {
               System.out.println("");
               System.out.println("Удалите следующие объекты из переменной среды CLASSPATH:");

               for (int i = 0; i < classpathRemovals.size(); i++)
               {
                  System.out.print("   ");
                  System.out.println((String)classpathRemovals.elementAt(i));
               }
            }
         }
      }

      catch (Exception e)
      {
         // Если при выполнении какой-либо операции возник сбой -
         // появляется сообщение об ошибке и текст исключительной ситуации.

         System.out.println("Установка/обновление не выполнены");
         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("  checkToolbox sourcePath targetPath");
         System.out.println("");
         System.out.println("Где");
         System.out.println("");
         System.out.println("   sourcePath = исходный путь файлов IBM Toolbox for Java");
         System.out.println("   targetPath = целевой путь файлов IBM Toolbox for Java");
         System.out.println("");
         System.out.println("Например:");
         System.out.println("");
         System.out.println("   checkToolbox http://mySystem/QIBM/ProdData/HTTP/Public/jt400/ d:\\jt400");
         System.out.println("");
         System.out.println("");
      }

      System.exit(0);
   }
}
```

Пример: Применение CommandPrompter

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

В этом разделе перечислены примеры программ, встречающиеся в документации по классам графического интерфейса IBM Toolbox for Java.

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
- Пример: Добавление ActionCompletedListener для обработки сообщений команды, выполняемой в системе iSeries
- 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
- Пример: Обработка всех созданных программой сообщений iSeries
- 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

- Пример: Создание объекта SpooledFileViewer для просмотра созданного ранее буферного файла iSeries

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

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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

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

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

```
///////////////////////////////////////////////////////////////////////////
//
// Пример работы с объектом VMessageList. Эта программа показывает
// 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

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

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

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

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("  система = сервер iSeries");
          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

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

```
///////////////////////////////////////////////////////////////////////
//
// Пример применения объекта IFSTextFileDocument. Эта программа
// демонстрирует работу с документом, связанным с текстовым
// 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

Класс AS400JDBCDataSourcePane представляет значения свойств объекта AS400JDBCDataSource. 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 ();
```

Пример: Применение класса VJobList для вывода списка заданий

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

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

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

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

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 система <пользователь>
//
//    (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("Применение: VPrinterOutputExample система <пользователь>");
            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

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

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта SQLQueryBuilderPane. Эта программа
// позволяет пользователю создавать запросы SQL.
//
// 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

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

```
////////////////////////////////////////////////////////////////////////
//
// Пример применения объекта SQLResultSetTablePane. Эта программа
// показывает содержимое таблицы. Пользователь может вводить любые
// операторы SQL с помощью класса SQLStatementDocument. Кроме того,
// пользователь может удалять сроки таблицы с помощью предназначенной
// для этого кнопки.
//
// 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 компоненту XPCML IBM Toolbox for Java.

- “Пример: Уплотнение существующего документа XPCML” на стр. 746
- “Пример: Уплотнение существующего документа XPCML с исходным кодом Java” на стр. 747
- “Пример: Создание объекта ProgramCallDocument с помощью уплотненного файла XPCML” на стр. 749
- “Пример: Получение результатов вызова программы в виде уплотненного файла XPCML” на стр. 750
- “Пример: Получение результатов вызова программы в виде файла XPCML” на стр. 741
- “Пример: Передача значений параметров в виде файла XPCML” на стр. 743
- “Пример: Передача массивов значений параметров в виде файла XPCML” на стр. 745
- “Пример: Преобразование документа PCML в документ XPCML” на стр. 436

Следующее заявление об отказе от гарантий относится ко всем примерам кода IBM Toolbox for Java:

Отказ от гарантий на предоставляемый код

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

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: Получение результатов вызова программы в виде файла XPCML

В приведенном ниже примере создается объект XPCML ProgramCallDocument и вызывается программа iSeries, результаты которой возвращаются в виде документа 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. В противном случае в теге XPCML <arrayOfStructParm> были бы перечислены 89 пользователей.

В массивах структур XPCML ограничивает каждый элемент structParm с помощью тега <struct_i>. Каждый тег <struct_i> указывает, что в нем заключен один элемент структуры типа autu0150. Атрибут индекса тега <struct_i> задает элемент массива для структуры.

Для массивов простых типов данных, таких, как arrayOfStringParm, arrayOfIntParm, и т.д., тег XPCML <i> содержит список элементов массива.

Документ XPCML qgyolaus.xpcml

```
<xpcml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="xpcml.xsd" version="4.0">

<!-- Исходный код XPCML для вызова API Открыть список пользователей с правами доступа (QGYOLAUS) -->
<!-- (QGYOLAUS) -->

  <!-- Формат AUTU0150 - Поддерживаются дополнительные форматы -->
  <struct name="autu0150">
    <stringParm name="name" length="10"/>
    <stringParm name="userOrGroup" length="1"/>
    <stringParm name="groupMembers" length="1"/>
    <stringParm name="description" length="50"/>
  </struct>

  <!-- Информационная структура списка (общая для всех API Открыть список... -->
  <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>

  <!-- Программа QGYOLAUS и список ее параметров для получения данных в формате
  <!-- AUTU0150 -->

  <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>     <!-- Конец массива имен пользователей -->
            <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

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

В приведенном ниже примере показано, как можно передать массив значений параметров с помощью данных массива structParm и массива объектов struct.

```
<?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>
```

Указание значений элементов с помощью атрибута index тегов <i> и <struct_i>

Задать значения элементов массива можно с помощью атрибута index тегов <i> и <struct_i>. 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'>
 <!-- Обратная ссылка на 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. Он содержит исходный документ XPCML, полученный уплотненный документ XPCML, код Java, в котором вызывается метод condenseXPCML(), и некоторые новые определения типов в расширенной схеме:

Исходный документ XPCML

```
      <!-- Полный документ XPCML -->
      <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>

       <!-- Ниже приведены дополнительные определения типов для каждого нового типа -->
       </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

Получить результаты вызова программы в виде уплотненного или неуплотненного файла XPCML можно одним и тем же способом. Для этого достаточно вызвать метод ProgramCallDocument.generateXPCML().

С помощью метода setXsdName() необходимо задать имя расширенной схемы, которая применяется методом gengrateXPCML() для создания атрибута noNamespaceSchemaLocation тега <xpcml> уплотненного файла 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>
```

Дополнительная информация о продукте IBM Toolbox for Java

Ниже приведен список Web-сайтов и разделов Information Center, содержащих дополнительную информацию о продукте IBM Toolbox for Java.

Ресурсы IBM Toolbox for Java

Дополнительная информация о IBM Toolbox for Java приведена на следующих сайтах:

- IBM Toolbox for Java и JTOpen : Содержит информацию о пакетах обслуживания, советы по повышению производительности, примеры программ и многое другое. Здесь же вы найдете файл .zip с этой информацией, включая javadocs.
- На странице IBM Toolbox for Java Frequently Asked Questions (FAQ) приведены ответы на вопросы о производительности, устранении неполадок, JDBC и другие вопросы.
- Форум IBM Toolbox for Java и JTOpen : позволяет эффективно обмениваться информацией с разработчиками, применяющими Java и IBM Toolbox for Java, а также создателями IBM Toolbox for Java.

Ресурсы IBM Toolbox for Java 2 Micro Edition

Дополнительная информация о продукте ToolboxME for iSeries и применении Java для беспроводных технологий приведена на следующих сайтах:

- Страница IBM Toolbox for Java и JTOpen содержит дополнительную информацию о ToolboxME for iSeries.
- 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)
- На странице Java Wireless Developer приведена различная техническая информация для разработчиков приложений на Java для беспроводных устройств.
- 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.
- На странице IBM alphaWorks Java приведены сведения о новых технологиях Java, а также загружаемые файлы 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.
- Foundation - Java, IBM eServer enablement Technical resources : содержит информацию о Java и о том, как деловые партнеры IBM применяют приложения на Java.

Java Naming and Directory Interface

- Java Naming and Directory Interface(TM) (JNDI) : Обзор JNDI, техническая информация, примеры и список доступных поставщиков услуг.
- На странице iSeries Directory Server (LDAP) приведены сведения о поддержке LDAP (простого протокола доступа к каталогам) в системе OS/400.

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 Foundation - XML, IBM eServer enablement Technical resources содержит информацию о XML и о том, как деловые партнеры IBM применяют эту технологию.
- Страница W3C XML содержит список технических ресурсов для разработчиков XML.
- На Web-сайте XML.com приведена информация об роли XML в современной компьютерной индустрии.
- На Web-сайте XML.org размещены 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

- Web-сайт IBM HTTP Server for iSeries содержит информацию, ресурсы и советы по применению IBM HTTP Server for iSeries.
- iSeries Access for Windows : Информация о продукте iSeries Access для Windows, включая файлы для загрузки, FAQ и ссылки на дополнительные сайты.
- IBM WebSphere Host On-Demand : Информация об эмуляторе на основе браузера, обеспечивающем поддержку эмуляции S/390, iSeries и DEC/Unix.
- На Web-сайте IBM Support and downloads размещен портал, посвященный поддержке аппаратного и программного обеспечения IBM.

Отказ от гарантий на предоставляемый код

Данный документ содержит примеры программного кода.

Фирма IBM предоставляет вам неисключительное право на использование всех этих примеров программного кода, на основе которых вы можете создавать собственные программы.

Все примеры кода приведены фирмой IBM исключительно для иллюстрации. Они не были тщательно и всесторонне протестированы. По этой причине фирма IBM не может гарантировать их надежность, удобство их обслуживания и отсутствие в них ошибок.

Все приведенные программы предоставляются на условиях ″КАК ЕСТЬ″ без каких-либо гарантий. В частности, не предоставляются никакие подразумеваемые гарантии соблюдения авторских прав, коммерческой ценности и пригодности для каких бы то ни было целей.

Условия загрузки и печати публикаций

Разрешение на использование выбранных для загрузки публикаций предоставляется в соответствии с следующими условиями и при подтверждении вашего с ними согласия.

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 данные публикации в рамках своей организации 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.

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 предоставляет гарантий на содержание этих публикаций. Публикации предоставляются на условиях ″как есть″, без каких-либо явных или подразумеваемых гарантий, включая, но не ограничиваясь этим, подразумеваемые гарантии коммерческой ценности или применения для каких-либо конкретных целей.

Авторские права на все материалы принадлежат IBM Corporation.

Загружая или печатая публикации с этого сайта, вы тем самым подтверждаете свое согласие с приведенными условиями.