BlackBerry Java Development Environment

Version 4.0

BlackBerry Application Developer Guide

Volume 1: Fundamentals

BlackBerry Java Development Environment Version 4.0 BlackBerry Application Developer Guide Volume 1: Fundamentals

Last modified: 26 November 2004

Part number: SWD_X_JDE(EN)-001.000

At the time of publication, this documentation complies with BlackBerry Java Development Environment Version 4.0.

© 2004 Research In Motion Limited. All rights reserved. The BlackBerry and RIM families of related marks, images and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, BlackBerry and 'Always On, Always Connected' are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

Microsoft, Windows and Outlook are registered trademarks of Microsoft Corporation in the United States and/or other countries. Java and Jini are trademarks of Sun Microsystems, Inc. in the U.S. and other countries. IBM, Lotus, and Domino are trademarks of International Business Machines Corporation in the United States. The Bluetooth® word mark and logos are owned by the Bluetooth SIG, Inc. and any use of such marks by Research In Motion is under license. All other brands, product names, company names, trademarks, and service marks are the properties of their respective owners.

The handheld and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D,445,428; D,433,460; D,416,256. Other patents are registered or pending in various countries around the world. Please visit www.rim.net/patents.shtml for a current listing of applicable patents.

This document is provided "as is" and Research In Motion Limited (RIM) assumes no responsibility for any typographical, technical, or other inaccuracies in this document. RIM reserves the right to periodically change information that is contained in this document; however, RIM makes no commitment to provide any such changes, updates, enhancements, or other additions to this document to you in a timely manner or at all. RIM MAKES NO REPRESENTATIONS, WARRANTIES, CONDITIONS, OR COVENANTS, EITHER EXPRESS OR IMPLIED (INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OF FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, MERCHANTABILITY, DURABILITY, TITLE, OR RELATED TO THE PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE REFERENCED HEREIN, OR PERFORMANCE OF ANY SERVICES REFERENCED HEREIN). IN CONNECTION WITH YOUR USE OF THIS DOCUMENTATION, NEITHER RIM NOR ITS AFFILIATED COMPANIES AND THEIR RESPECTIVE DIRECTORS, OFFICERS, EMPLOYEES, OR CONSULTANTS SHALL BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER BE THEY DIRECT, ECONOMIC, COMMERCIAL, SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY, OR INDIRECT DAMAGES, EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, INCLUDING, WITHOUT LIMITATION, LOSS OF BUSINESS REVENUE OR EARNINGS, LOST DATA, DAMAGES CAUSED BY DELAYS, LOST PROFITS, OR A FAILURE TO REALIZE EXPECTED SAVINGS.

This document might contain references to third-party sources of information and/or third-party web sites ("Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation, the content, accuracy, copyright compliance, legality, decency, links, or any other aspect of Third-Party Information. The inclusion of Third-Party Information in this document does not imply endorsement by RIM of the third party in any way. Any dealings with third parties, including, without limitation, compliance with applicable licenses, and terms and conditions are solely between you and the third party. RIM shall not be responsible or liable for any part of such dealings.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Research In Motion UK Limited
Centrum House, 36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

Published in Canada

Contents

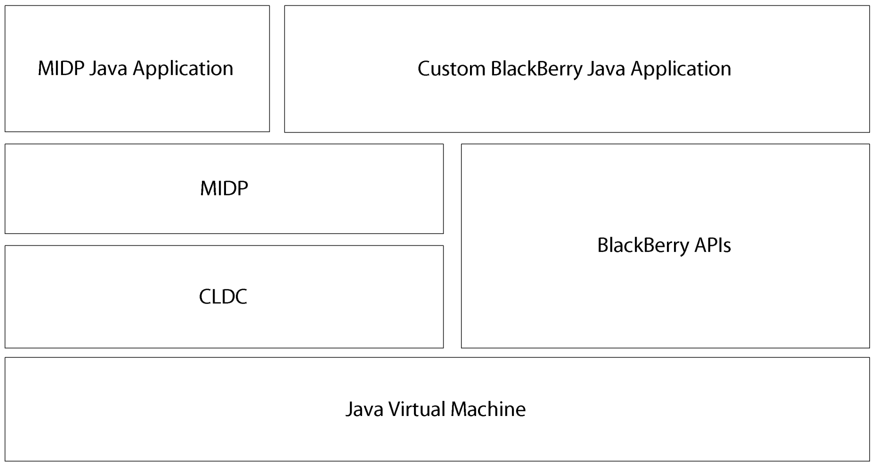
BlackBerry APIs

Using BlackBerry APIs

The BlackBerry® Java Development Environment (JDE) provides a complete set of APIs and tools for you to develop Java™ applications that run on BlackBerry Wireless Handhelds™. Applications developed using version 4.0 of the JDE run on handhelds running BlackBerry Handheld Software version 3.8 or 4.0.

BlackBerry handhelds include a J2ME runtime environment that is based on the CLDC 1.1 and MIDP 2.0 specifications. BlackBerry API extensions provide additional capabilities and tighter integration with BlackBerry handhelds.

You can use both CLDC/MIDP APIs and BlackBerry APIs in your application. To enable applications to run on any JTWI-enabled device, write standard MIDP applications using only the CLDC and MIDP APIs.



BlackBerry handheld software components

To view the *API reference*, on the taskbar, click Start > Programs > Research In Motion > BlackBerry Java Development Environment 4.0 > API Reference.

BlackBerry APIs

The BlackBerry APIs provide access to BlackBerry features for user interfaces, localization, networking, and other capabilities.

Note: Access to additional APIs for features, such as advanced cryptography, synchronization, and messaging, is restricted. To use these APIs, you must receive express written permission from an authorized signatory of Research In Motion. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.

BlackBerry API package	Description
`net.rim.blackberry.api.browser`	This package enables applications to invoke the BlackBerry Browser. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.invoke`	This package enables applications to invoke BlackBerry applications, such as tasks, messages, MemoPad and phone. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.mail`	This package defines the functionality necessary to convert the components of internal RIM email system objects into portable objects that are compatible with the mail API. It also provides functionality for sending, receiving and accessing email messages. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.mail.event`	This package defines messaging events and listener interfaces to manage mail events. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.menuitem`	This package enables you to add custom menu items to BlackBerry applications, such as the address book, calendar, and messages. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.options`	This package enables you to add items to the handheld options. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.pdap`	This package enables applications to interact with BlackBerry personal information management (PIM) applications, including address book, tasks, and calendar. Most of the same functionality is provided by the MIDP package `javax.microedition.pim`. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.phone`	This package provides access to advanced features of the phone application. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.blackberry.api.phone.phonelogs`	This package provides access to the phone call history. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.device.api.bluetooth`	This package enables BlackBerry applications to communicate with Bluetooth® wireless technology enabled devices on a Bluetooth serial port connection. See " Using Bluetooth serial port connections" on page 90 for more information.
`net.rim.device.api.browser.field`	This package enables applications to display a browser field within their user interface. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.device.api.browser.plugin`	This package enables you to add support for additional MIME types to the BlackBerry Browser. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.device.api.collection` `net.rim.device.api.collection.util`	This package defines interfaces and utility classes for managing data collections. See " Collections" on page 9 for more information.
`net.rim.device.api.compress`	This package provides utilities for compressing data, in both GZip and Zlib formats. Decompression is not currently implemented; handhelds can write out valid GZip and Zlib files, but the content is not compressed. See " Compression" on page 9 for more information.
`net.rim.device.api.i18n`	This package provides classes to support the localization of applications on BlackBerry handhelds. See " Localizing applications" on page 109 for more information.
`net.rim.device.api.io`	This package provides a library of custom BlackBerry classes for managing data input and output.
`net.rim.device.api.mime`	This package provides classes for working with streams of MIME-encoded data.
`net.rim.device.api.notification`	This package provides methods to trigger event notifications and respond to system-wide and application-specific events. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.

BlackBerry API package	Description
`net.rim.device.api.servicebook`	This package enables applications to add, delete, and access service book entries. See the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information.
`net.rim.device.api.system`	This package provides access to system-level functionality, including event listeners for the handheld keyboard and trackwheel, image creation and support, and application control.
`net.rim.device.api.ui`	This package provides enhanced functionality to control the BlackBerry user interface, including screen and field layout managers, field type support, and focus, scroll, and change listeners. See " User interface APIs" on page 29 for more information.
`net.rim.device.api.ui.component`	This package provides a library of interface components for creating UI applications. See the See " Displaying UI components" on page 29 for more information.
`net.rim.device.api.ui.container`	This package provides a library of interface manager components for creating UI applications. See " Managing UI components" on page 38 for more information.
`net.rim.device.api.ui.text`	This package provides classes to filter text strings containing various kinds of data, such as phone numbers or URLs.
`net.rim.device.api.util`	This package provides utility methods and interfaces that are useful within the context of the handheld system, including classes for arrays, hash tables, and string matching.

CLDC APIs

CLDC API package	Description
`java.io`	This package provides for system input and output through data streams.
`java.lang`	This package provides classes that are fundamental to the design of the Java programming language.
`java.lang.ref`	This package provides reference-object classes, which support a limited degree of interaction with the garbage collector.
`java.util`	This package contains the collection classes, date and time facilities, and miscellaneous utility classes.
`javax.microedition.io`	This package contains classes for generic connections.

MIDP APIs

MIDP API package	Description
`javax.microedition.lcdui`	This package contains the MIDP UI API, which provides a set of features for implementation of user interfaces for MIDP applications.
`javax.microedition.lcdui.game`	This package contains classes that enable the development of rich gaming content for wireless devices.
`javax.microedition.midlet`	This package defines Mobile Information Device Profile applications and the interactions between the application and the environment in which the application runs.
`javax.microedition.pki`	This package defines certificates that are used to authenticate information for secure connections.
`javax.microedition.rms`	This package provides a mechanism for MIDlets to store and retrieve persistent data.

PDAP APIs

MIDP API package	Description
`javax.microedition.pim`	This package provides a standard mechanism for accessing PIM information.

Using Java on BlackBerry handhelds

Source code is compiled and packaged into .cod files. The .cod files are loaded onto BlackBerry handhelds and run by their virtual machines (VM).

Note: The .cod name is limited to 128 bytes.

The BlackBerry JDE uses a *split VM* architecture, as described in the CLDC specification. To reduce the amount of memory and processing power that is required on handhelds, part of the class loading process, called *preverification*, occurs before the Java code is loaded onto the handheld. The IDE preverifies source files automatically, before packaging them into .cod files. The VM performs the remainder of verification during class loading onto handhelds.

Restrictions

The BlackBerry Wireless Handheld VM has the following restrictions, as specified by CLDC 1.1:

- no object finalization
- no `java.lang.Error` class hierarchy
- no user class loading
- no reflection, therefore no support for Remote Method Invocation (RMI) or Jini™ network technology
- no native methods
- no `Runtime.exec()` for running external processes

Multithreading

The BlackBerry Java environment provides a true multithreading environment for running applications. This enables multiple applications to run simultaneously, events to broadcast to multiple applications, and long operations or listener threads to run in the background.

Persistent storage

Data stored in flash memory persists between handheld resets. Store data on the handheld in one of two ways:

- using MIDP record stores
- using the BlackBerry persistence model

See "Storing persistent data" on page 79 of the *BlackBerry Application Developer Guide Volume 2: Advanced Topics* for more information on storing persistent data using the BlackBerry APIs.

Network communication

The BlackBerry JDE implements network communication according to the MIDP 2.0 specification. It provides a variety of connectivity options, including the ability to securely connect behind corporate firewalls using proxied HTTP connections.

The BlackBerry JDE provides the following connection types:

- stream connections (`StreamConnection` interface), including:
  - HTTP connections (`HttpConnection` interface)
  - HTTPS connections (`HttpsConnection` interface)
  - socket connections (`SocketConnection` interface)
  - secure socket connections (`SecureConnection` interface)
  - serial connections to a communication port on the handheld (`CommConnection` interface)
- datagram connections (`DatagramConnection` interface), including:
  - UDP datagram connections (`UDPDatagramConnection` interface)

The `javax.microedition.io.PushRegistry` class maintains a list of inbound connections to the handheld.

See " Connecting to networks" on page 82 for more information. See `Connector` in the *API Reference* for detailed information on opening each of the connection types.

Streams

The BlackBerry JDE provides the standard interfaces and classes for streams that are included in the CLDC `java.io` package.

MIME encoding

The BlackBerry JDE provides `MIMEInputStream` and `MIMEOutputStream` classes for reading and writing a MIME-encoded data stream.

Class	Description
`MIMEInputStream`	This class implements a stream that reads a MIME message, and then formats and parses the message into its parts according to the MIME standard.
`MIMEOutputStream`	This class implements an output stream that can format output into parts according to the MIME standard. This class does not perform the actual data encoding, so you must encode data before writing it to this stream.

Compression

The BlackBerry JDE provides classes, in the `net.rim.device.api.compress` package, for reading data streams compressed using either the ZLib or GZip formats. These classes behave much like the corresponding classes in the `java.util.zip` package in J2SE.

Decompression is not currently implemented; handhelds can write out valid GZip and Zlib files, but the content of such files is not compressed.

Collections

The BlackBerry JDE provides a set of interfaces and utility classes for managing collections on the handheld.

The `net.rim.device.api.collection` package includes interfaces that define various types of collections, such as lists, sets, and maps, for specific types of data. These interfaces define capabilities similar to the `List`, `Set`, and `Map` interfaces in the J2SE Collections Framework.

Implement these interfaces in your own classes, or use the utility classes that are provided in the `net.rim.device.api.collection.util` package.

Vectors

The standard `java.util.Vector` class implements a resizeable array of objects. The BlackBerry JDE also provides convenience classes, such as `net.rim.device.api.util.IntVector` and `net.rim.device.api.util.ByteVector` for working with arrays of primitive types.

For large arrays of data (more than 10 or 15 KB), the BlackBerry JDE provides the `BigVector`, `BigLongVector`, and `BigIntVector` classes in `net.rim.device.api.collection.util`. These classes are like normal vectors, except that they are optimized for inserting items at any location. In contrast, if you make random changes using standard large vectors, large amounts of data move between flash memory and RAM.

Lists

The BlackBerry JDE provides classes in the `net.rim.device.api.collection.util` package to manage lists of elements:.

Class	Description
`SortedReadableList` and `UnsortedReadableList`	Use these classes to maintain sorted or unsorted lists of elements. The `SortedReadableList` class requires you to use a comparator object to sort the items in the list; each item that you add to the list must be recognized as valid by this comparator.
`IntSortedReadableList` and `LongSortedReadableList`	Use these classes to automatically sort lists of integers or elements that are associated with long integer keys.
`BortedReadableList` and `BigUnsortedReadableList`	Use these classes to store large collections of data (more than 10 or 15 KB). These classes do not store data in an array, so you can make random changes to large data collections more efficiently.
`ReadableListCombiner`	Use this class to combine two or more `ReadableList` objects and present them as a single `ReadableList`.
`ReadableListUtil`	Use this class, which provides utility methods such as `getAt()` and `getIndex()`, to retrieve data from readable lists.

Hash tables

In addition to the standard `java.util.Hashtable` class that the CLDC provides, the BlackBerry JDE includes a specialized `net.rim.device.api.collection.util. LongHashtableCollection` class, which provides a hash table collection that uses long integers as keys. With a `LongHashtableCollection` object, write operations occur as a map (using a key-element pair) and read operations occur as a map or as a set (retrieving the elements in the collection as an array).

Event listeners

Event listener interfaces are divided by event type. Each application registers to receive specific types of events. The application event queue then dispatches events to the appropriate listeners.

Applications can implement the appropriate listener interfaces or override the listener methods on the various `Screen` objects. Most applications implement the `KeyListener` and `TrackwheelListener` interfaces and register the listeners to receive keyboard and trackwheel events. The keyboard and trackwheel are the primary means by which users interact with applications.

The following event listeners are located in the `net.rim.device.api.system` package:

Listener Interface	Type of event
`AlertListener`	Implement this interface to listen for alert events.
`BluetoothSerialPortListener`	Implement this interface to listen for Bluetooth serial port events, such as the opening of a Bluetooth serial port connection either as a server or client.
`GlobalEventListener`	Implement this interface to listen for global events that are broadcast to all applications.
`HolsterListener`	Implement this interface to listen for holster events, such as the handheld being placed into or removed from the holster.
`IOPortListener`	Implement this interface to listen for I/O port events.
`KeyListener`	Implement this interface to listen for keyboard events, such as the user pressing or releasing a key.
`RealTimeClockListener`	Implement this interface to listen for real-time clock events, such as the clock being updated.
`SerialPortListener`	Implement this interface to listen for serial port events, such as a change in the status of data being sent to the serial port connection, for handhelds that are connected to the computer serial port.
`SystemListener`	Implement this interface to listen for system events, such as changes to battery status and power.
`TrackwheelListener`	Implement this interface to listen for trackwheel events, such as the user clicking the trackwheel.
`USBPortListener`	Implement this interface to listen for USB port events, such as the status of data being sent to the USB port connection, for handhelds that are connected to the computer USB port.

System capabilities

The classes in the `net.rim.device.api.system` package provide access to the Java VM and system-wide resources on the handheld.

Retrieve radio information

The `RadioInfo` class provides access to information on the status of the handheld radio.

Retrieve handheld information

The `DeviceInfo` class provides access to the following information on handhelds:

- battery power and status
- device ID
- idle time
- platform version

Alert users

The `Alert` class enables your application to notify users when an event, such as the arrival of a new message, occurs.

Monitor memory usage

Use the static methods provided by the `Memory` class to retrieve statistics on VM memory usage.

Some of the utility methods in the `Memory` class return a `MemoryStats` object. Use the utility methods the `MemoryStats` class provides to retrieve detailed information on the memory and storage that is available on the handheld.

Log events

The `EventLogger` class enables applications to store event logs in the persistent store. The handheld maintains an event queue so that, when the log is full, the oldest events are deleted as new events are added. Users can view the system event log on the handheld by holding the **Alt** key and typing **lglg**.

Utilities

The BlackBerry JDE provides a set of utilities in the `net.rim.device.api.util` package. Many of these classes provide similar capabilities to utilities in J2SE.

- The `Comparator` interface, similar to the one provided in J2SE, defines methods that impose order on a collection of objects.
- The `Arrays` class provides methods for working with arrays, such as sorting and searching, and viewing arrays as lists.
- The `BitSet` class maintains a collection of bits.

The `net.rim.device.api.util` package includes several convenient classes for managing specific types of data collections, including vectors, hash tables, maps, and stacks.

Application control

Application control enables system administrators to perform the following actions:

- exclude applications from existing on handhelds
- limit internal connections (connections behind a corporate firewall)
- limit external connections
- limit local connections (serial, infrared, and USB connections)
- limit access to the key store
- limit access to particular APIs

See the *BlackBerry Enterprise Server Handheld Management Guide* for more information on application control.

APIs with limited access

Applications that use the following restricted APIs can load on handhelds, but throw a `ControlledAccessException` at runtime if they access an API not permitted under application control.

API	Default value without IT policy	Default value with IT policy
Bluetooth API (`net.rim.device.api.bluetooth`)	allowed	allowed
Mail API (`net.rim.blackberry.api.mail`)	allowed	allowed
PIM API (`net.rim.blackberry.api.pdap`)	allowed	allowed

API	Default value without IT policy	Default value with IT policy
Phone API and invocation API (used to invoke the phone application) (`net.rim.blackberry.api.phone` and `net.rim.blackberry.api.invoke`)	initiating allowed (no prompt by default)	initiating allowed (user prompted by default)
Notification API (`net.rim.device.api.notification`)	allowed	not permitted
HTTP Filter API (`net.rim.device.api.io.http`)	allowed	not permitted

Writing BlackBerry Java applications

Application management

When the handheld starts, the VM loads an application manager, which manages all Java applications on the handheld. The application manager functions as the central dispatcher of operating system events for other Java applications.

Applications that provide a user interface extend the `net.rim.device.api.ui.UiApplication` class. This class provides methods for applications to register event listeners, manage threads, and manage UI components.

BlackBerry applications start at `main()`. When an application starts, its `main()` thread calls `enterEventDispatcher()` to start handling events. This thread runs all drawing and event-handling code, and waits for events on the application queue.

When the application manager receives an event, it copies the event to the appropriate queues, which enables the application manager to direct messages to certain programs. For example, only the foreground application receives user input messages.

Writing a sample application

Extend the UiApplication base class

Each application that provides a user interface must extend the `UiApplication` base class. The `UiApplication` class defines methods for applications to establish an event thread, and display and maintain `Screen` objects.

Define main()

In `main()`, create a new object for the application. Call `enterEventDispatcher()` for the application to enter the event thread and start processing messages.

```
public static void main(String[] args) {
   HelloWorld theApp = new HelloWorld();
   theApp.enterEventDispatcher();
}
```

Define a constructor

Define the default constructor for your application. The default constructor invokes `UiApplication.pushScreen()` to display the screen that appears when the application starts. In this example, the screen is a new instance of `HelloWorldScreen`, which you define in the following section:

```
public HelloWorld() {
   pushScreen(new HelloWorldScreen());
}
```

Define the main screen

To define the main screen of the application UI, extend the `MainScreen` class. The `MainScreen` class is a subclass of `Screen`, which implements the `TrackwheelListener` and `KeyboardListener` interfaces. These interfaces receive and respond to user interaction. If you extend the `Screen` class, or one of its subclasses, you do not have to implement the `TrackwheelListener` and `KeyboardListener` interfaces.

Your class should override at least two of the `MainScreen` methods, the default constructor and `onClose()`.

In this example, the constructor invokes the `MainScreen` constructor. By default, `MainScreen` provides the following features:

- A default menu with a **Close** menu item.
- Default close action when the user clicks **Close** or presses **Escape**. To provide custom behavior, such as displaying a dialog box alert, when the user clicks the **Close** menu item or presses the **Escape** button, override `onClose()`.
- An instance of `RichTextField`, a read-only rich text field that can receive focus. See " Provide screen navigation" on page 30 for more information on adding UI components to a screen.
- A context menu with a **Select** menu item. See " Creating custom context menus" on page 49 for more information.

Code example

The following example creates a screen that contains a rich text field. When the rich text field receives focus, the menu includes a **Close** item and a **Select** context menu item.

Example: HelloWorld.java

```
/**
 * HelloWorld.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.helloworld;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

public class HelloWorld extends UiApplication {
    public static void main(String[] args) {
        HelloWorld theApp = new HelloWorld();
        theApp.enterEventDispatcher();
    }
    public HelloWorld() {
        pushScreen(new HelloWorldScreen());
    }
}

final class HelloWorldScreen extends MainScreen {
    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField("HelloWorld Sample", LabelField.ELLIPSIS
            | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Hello World!"));
    }

    public boolean onClose() {
        Dialog.alert("Goodbye!");
        System.exit(0);
        return true;
    }
}
```

Reusing common code

Abstract base classes enable you to implement and reuse common functionality across multiple classes. Each application can extend a single base class.

In the IDE, add the base class to a library project. Create separate projects for each application and define a dependency on the library project.

Code example

The sample applications in this guide extend the `BaseApp` class, which implements the following functionality:

- extends the `UiApplication` class
- implements the `KeyListener` and `TrackwheelListener` interfaces
- defines variables, such as common menu items
- defines a method to create an application menu
- defines a method for menu selection
- defines an abstract method to exit the application

Example: BaseApp.java

```
/*
 * BaseApp.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.baseapp;

import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;

public abstract class BaseApp extends UiApplication implements BaseAppResource,
   KeyListener, TrackwheelListener {
    private MenuItem _closeItem;
    private static ResourceBundle _resources =
    ResourceBundle.getBundle(BUNDLE_ID, BUNDLE_NAME);
    /* Constructor for the abstract base class. */
    public BaseApp() {
        _closeItem = new MenuItem(_resources, MENUITEM_CLOSE, 200000, 10) {
            public void run() {
                onExit();
                System.exit(0);
            }
        };
    }
    /* Override this method to add custom menu items. */
    protected void makeMenu( Menu menu, int instance) {
        Field focus = UiApplication.getUiApplication().
        getActiveScreen().getLeafFieldWithFocus();
        if(focus != null) {
            ContextMenu contextMenu = focus.getContextMenu();
            if( !contextMenu.isEmpty()) {
                menu.add(contextMenu);
                menu.addSeparator();
            }
        }
        menu.add(_closeItem);
    }
    /* Invoked when the trackwheel is clicked. */
    public boolean trackwheelClick( int status, int time ) {
        Menu menu = new Menu();
        makeMenu( menu, 0);
        menu.show();
        return true;
    }
    /* Invoked when the trackwheel is released. */
    public boolean trackwheelUnclick( int status, int time ) {
        return false;
    }
    /* Invoked when the trackwheel is rolled. */
    public boolean trackwheelRoll(int amount, int status, int time) {
            return false;
        }
        public boolean keyChar(char key, int status, int time) {
            /* Intercept the ESC key and exit the application. */
            boolean retval = false;
            switch (key) {
                case Characters.ESCAPE:
                onExit();
                System.exit(0);
                retval = true;
                break;
            }
            return retval;
        }
        /* Implementation of KeyListener.keyDown(). */
        public boolean keyDown(int keycode, int time) {
            return false;
        }
        /* Implementation of KeyListener.keyRepeat(). */
        public boolean keyRepeat(int keycode, int time) {
            return false;
        }
        /* Implementation of KeyListener.keyStatus(). */
        public boolean keyStatus(int keycode, int time) {
            return false;
        }
        /* Implementation of KeyListener.keyUp(). */
        public boolean keyUp(int keycode, int time) {
            return false;
        }
        protected abstract void onExit();
    }
```

Using the IDE

Writing applications

To write applications, use the IDE that is included with the BlackBerry JDE.

Create the workspace in a main folder, and then create subfolders for each project. Save all source files in the appropriate project folder and save other files, such as images and resource files, in separate subfolders. As with all Java programs, create a folder structure for your source code that matches the package hierarchy that you use for your classes.

Create a workspace

1. In the IDE, on the **File** menu, click **New Workspace**.
2. In the **Workspace name** field, type a name without a file name extension.
3. In the **Create in this directory** field, type a folder.

4. Click **OK**.

Create a project

1. In the IDE, on the **Project** menu, click **Create New Project**.
2. In the **Project name** field, type a project name without a file name extension.
3. In the **Create project in this directory** field, type the folder in which to create the project file.
4. Click **OK**.
5. In the workspace Files pane, double-click the project name to set the project properties.

   See the *IDE Online Help* for information on project properties.

Creating source files

1. In the IDE, on the **File** menu, click **New**.
2. In the **Source file name** field, type a file name with the .java file name extension.
3. In the **Create source file in this directory** field, type a folder name.
4. Click **OK**.
5. In the editor pane, right-click the file, and then click **Insert into project**.
6. Select a project, and then click **OK**.

Building projects

When you build a project, the IDE compiles your source files into Java bytecode, performs preverification, and then packages the classes into a .cod file.

Note: In J2ME, bytecode verification is divided into two stages. The compiled code is preverified before it is loaded onto the handheld, so the handheld only has to perform basic verification as classes are loaded. The IDE performs preverification automatically when it builds projects.

When you build a project, the IDE also builds any libraries on which the project depends, if necessary.

Action	Procedure	Additional information
Build all projects	▸ On the **Build** menu, click **Build All**.	To exclude a project, set the project properties.
Build all active projects	▸ On the **Build** menu, click **Build**.	In the workspace, active project names appear in bold. To change which projects are active, on the **Project** menu, click **Set Active Projects**.
Build a single project	1. Select a project. 2. On the **Build** menu, click **Build Selected**.	–
Create a workspace makefile	▸ On the **Build** menu, click **Generate Makefile and Resources**.	–

By default, the compiled .cod file uses the project name. To change this name double-click the project file, click the **Build** tab, and type the output file name.

Obfuscate applications

The IDE compiler provides a level of obfuscation in .cod files by default. For example, the IDE compiler automatically obfuscates internal, private members and removes debugging information.

1. Compile a .jar file for your application.
2. Use standard tools to obfuscate the .jar file.

3. In the IDE, create a new project.
4. Add the obfuscated .jar file to the project.
5. Build the project.

Generate javadocs

Use the IDE javadocs editor macro to facilitate adding javadocs comments to code.

Once enabled, if you type `/**` on any line preceding a function declaration, the IDE generates the following comment:

```
/**
* <description>.
* @param menu <description>.
* @param instance <description>.
* @return <description>.
*/
```

If you type `/**` on any other line, the IDE generates the following comment:

```
/**
* <description>.
*/
```

The IDE also preloads "`<description>`" as your search string, searches for the first instance, and selects that instance. This enables you to type a description, and then press F3 to move to subsequent parameters.

Because the javadocs macro relies on parsing the browsing information, add javadocs only after you perform a successful build. If your file contains a syntax error above where you are trying to insert javadocs, the macro does not pick up the function declaration.

Add a new editor macro

1. On the **Edit** menu, click **Preferences**.
2. Click the **Editor** tab and then click the **Macros** button.
3. From the **When I type** drop-down list, select `/**`. Type **@javadoc** in the **Replace it with** text box.
4. Type `/**` on the same line or the line immediately preceding each function declaration. For example, in the following code fragment you would place your cursor at the beginning of the word " protected" and type `/**`.

   ```
   /** protected int makeMenu(Menu menu, int instance) { ... }
   ```

Programming guidelines

Writing efficient code

Use local variables

Use local variables whenever possible. Access to local variables is more efficient than access to class members.

Use shorthand for evaluating Boolean conditions

Instead of unnecessarily evaluating a Boolean condition as shown in the first example, use shorthand as shown in the second. The resulting compiled code is shorter.

```
// Avoid this
if( boolean_expression == true ) {
   return true;
} else {
   return false;
}
// Do this
return( boolean_expression );
```

Make classes final

When you create code libraries, mark classes as `final` if you know that they will never be extended. The presence of the `final` keyword enables the compiler to generate more efficient code.

Note: By default, the BlackBerry JDE compiler marks any classes that are not extended in an application .cod file as final .

Use int instead of long

In Java, a `long` is a 64-bit integer. Because BlackBerry handhelds use a 32-bit processor, operations run two to four times faster if you use an `int` instead of a `long`.

Avoid garbage collection

Avoid calling `System.gc()` to perform garbage collection. This operation takes too much time, especially on handhelds with limited available memory. Let the VM collect garbage.

Using static variables for Strings

When defining static fields (also called *class fields*) of type `String`, you can increase program speed by using static variables (not `final`) instead of constants (`final`). The opposite is true for primitive data types, such as `int`.

For example, you might create a `String` object as follows:

```
private static final String x = "example";
```

For this static constant (denoted by the `final` keyword), a temporary `String` instance is created each time that you use the constant. The compiler eliminates "`x`" and replaces it with the string " example" in the bytecode, so that the VM performs a hash table lookup each that time you reference "`x`".

In contrast, for a static variable (no `final` keyword), the string is created once. The VM performs the hash table lookup only when it initializes " x" , so access is faster.

Note: You can use public constants (that is, `final` fields), but mark variables as `private`.

Avoid Object.getClass()

Avoid using `Object.getClass()` because it is not efficient and produces garbage (the `Class` object) that is never collected.

Avoid using the `example.class` literal. This literal generates `Class.forName("Example")`.

Avoid the String(String) constructor

Avoid using the `java.lang.String(String)` constructor because it creates an unnecessary `String` object that is a copy of the string that is provided as a parameter. Because `String` objects cannot be modified after they are created, copies are not typically necessary.

Note: The compiler generates a warning when you use the string constructor.

```
String str = new String("abc"); // avoid
String str = new String("found " + n + " items"); // avoid
```

In Java programs, each quoted string is implemented as an instance of the `java.lang.String` class. In other words, you create a `String` by writing code as in the following example:

```
String str = "abc"; // prefer
String str = "found " + n + " items"; // prefer
```

Write efficient loops

Factor loop-invariant code out of a loop.

```
// avoid
for( int i = 0; i < vector.size(); i++ ) {
    ...
}
```

This code results a call to `vector.size()` each time through the loop, which is inefficient. If your container is likely to have more than one element, it is faster to assign the size to a local variable. The following example factors out loop-invariant code:

```
// prefer
int size = vector.size();
for( int i = 0; i < size; ++i ) {
   ...
}
```

Alternatively, if the order in which you iterate through items is not important, you can iterate backward to avoid the extra local on the stack and to make the comparison faster.

```
for( int i = vector.size() - 1; i >= 0; --i ) {
   ...
}
```

Optimize subexpressions

If you use the same expression twice, do not rely on the compiler to optimize it for you. Use a local variable, as in the following example:

```
one( i+1 ); two( i+1 ); // avoid
int tmp = i+1; one( tmp ); two( tmp ); // prefer
```

Optimize division operations

Division operations can be slow on the handhelds because the processor does not have a hardware divide instruction.

When your code divides a positive number by two, use shift right by one (`>> 1`) instead. Use the “shift right" (>>) only when you know that you are dealing with a positive value.

```
midpoint = width / 2; //avoid
int = width >> 1; //prefer
```

Avoid java.util.Enumeration

Avoid using `java.util.Enumeration` objects unless you want to hide data (in other words, to return an enumeration of data instead of the data itself).

```
// avoid
for (Enumeration e = v.elements(); e.hasMoreElements();) {
   o = e.nextElement();
   ...
}
```

Asking a vector or hash table for an `Enumeration` object is slow and creates unnecessary garbage. Instead, iterate the elements yourself, as in the following example:

```
// prefer
for( int i = v.size() - 1; i >=0; --i ) {
   o = v.elementAt( i );
   ...
}
```

If the vector might be modified by another thread, synchronize the iteration as in the following example:

```
synchronized( v ) {
   for( int i = v.size() - 1; i >=0; --i ) {
      o = v.elementAt( i );
      ...
   }
}
```

Note: J2SE uses an `Iterator` object for similar operations, but iterator are not available in J2ME.

Perform casts using instanceof

Use `instanceof` to evaluate whether a cast succeeds instead of catching a `ClassCastException`.

```
// avoid
try {
   (String)x.whatever();
} catch( ClassCastException e ) {
   ...
}
// prefer
if( x instanceof String ) {
   (String)x.whatever();
} else {
   ...
}
```

Using `instanceof` is faster than using a `try/catch` block. Use the `try/catch` block only when a cast failure is an exceptional circumstance.

The BlackBerry JDE compiler and VM are optimized to perform only one class check in the first block of code. Perform the cast immediately following the branch that is determined by an `instanceof` check.

For example, the compiler can optimize the first example, but not the second:

```
// prefer
if ( a instanceof <type> ) {
   <type> instance = (<type>)a;
   x.method(instance);
   instance.method(x, y, z);
}

// avoid
if( a instanceof <type> ) {
   x.method( (<type>)a );
}
```

Evaluate conditions using instanceof

To produce smaller and faster code, if you evaluate a condition using `instanceof`, do not evaluate explicitly whether the variable is null. The condition `e instanceof <type>` evaluates to `false` if " e" is `null`.

```
// avoid
if( e != null && e instanceof ExampleClass ) { ... }
if( e == null || ! ( e instanceof ExampleClass) ) { ... }
// prefer
if( e instanceof ExampleClass ) { ... }
if( ! ( e instanceof ExampleClass ) ) { ... }
```

Reducing code size

Follow these guidelines when you write applications to reduce the size of the compiled code.

Set appropriate access

When you create code libraries, using the appropriate access modifiers for fields and methods significantly reduces the size of your compiled code. In particular, perform the following actions:

- Declare fields as `private` whenever possible. In addition to being good coding practice, this enables the compiler to optimize the .cod file.
- When possible, use the default (package) access instead of public access (that is, omit the `public` and `protected` keywords).

Avoid creating interfaces

When you create API libraries, avoid creating interfaces unless you foresee multiple implementations of the API. Interfaces produce larger, slower code.

Use static inner classes

When you use an inner class to hide one class inside another, but the inner class does not reference the outer class object, declare the inner class as `static`. This action suppresses the creation of a reference to the outer class.

For example, the following code requires a reference to the outer class object:

```
// avoid
class outer {
   int i;
   class inner {
      inner() {}
      int example() { return i; }
   }
}
```

In contrast, the following code only defines the scope of the inner class name:

```
// prefer
class outer {
   static class inner {
   ...
   }
}
```

The previous example is a shorter version of the following code:

```
class outer {
   ...
}
class outer$inner {
   ...
}
```

Only use a non-static inner class when you need access to data in the outer class from within methods of the inner class. If you use an inner class for name scoping, make it `static`.

Avoid unnecessary initialization

Avoid unnecessarily initializing fields in classes, where fields have default values. If you do not initialize a field in a class, it is initialized automatically using the following default values:

- an object reference is initialized to `null`
- int, byte, or long is initialized to 0
- Boolean is initialized to `false`

For example, there is no difference between the following code fragments:

```
// avoid
class BadExample {
   private int fieldsCount = 0; //avoid
   private Field _fieldWithFocus = null; //avoid
   private boolean _validLayout = false; //avoid
}
```

```
// prefer
class BetterExample {
   private int fieldsCount; //prefer
   private Field _fieldWithFocus; //prefer
   private boolean _validLayout; //prefer
}
```

Note: You must explicitly initialize local variables in a method.

Import individual classes

Applications that use only a small number of classes from a package should import the individual classes rather than the entire library.

```
// avoid
import net.rim.blackberry.api.browser.*;
// prefer
import net.rim.blackberry.api.browser.Browser;
```

Using time on BlackBerry handhelds

In time-sensitive applications, do not depend on time zones for anything other than displaying the local time to the user.

Handheld clock

The BlackBerry Wireless Handheld operating system calculates absolute time as milliseconds since midnight, January 1, 1970 Universal Time Coordinate (UTC). Time is typically measured in either CPU ticks or milliseconds.

System time zone changes

If you are caching a time-sensitive object for performance reasons, remember that the system time zone can change on the handheld.

When the time zone changes, the system sends out a global event message to the applications. To receive this event, implement the `GlobalEventListener` interface, including `eventOccurred()`, and register the listener by invoking `Application.addGlobalEventListener()`.

```
public void eventOccurred( long guid, int data0, int data1, Object object0,
   Object object1 ) {
   if( guid == DateTimeUtilities.GUID_TIMEZONE_CHANGED ) {
      _cal.setTimeZone( TimeZone.getDefault() );
   }
}
```

Recommended practices

Use multithreading

Make effective use of the multithreading capabilities of the BlackBerry operating system. In particular, always create a new thread for network connections or other lengthy operations (more than one-tenth of a second). Use background threads for listeners or other processes that run in the background when the application starts.

Minimize memory use

To minimize runtime memory, use the following guidelines:

- Use primitive types (such as `int` or `Boolean`) instead of objects (such as `String` or `Integer`).
- Do not depend entirely on the garbage collector. Avoid creating many objects quickly. Set object references to null when you have finished using them. Reuse objects as much as possible.
- Move heavy processing to the server. For example, perform data filtering or sorting before sending data to the handheld.

Avoid returning null

If you are writing a public method that returns an object, it should return `null` only under the following conditions:

- a null is expected during normal program operation
- the javadoc `@return` parameter states that null is a possible return value

If a `null` return value is not normally expected, then the method should throw an appropriate exception, which forces the caller to deal explicitly with the problem. The caller is not expected to check for a null return value, unless the documentation says otherwise.

Avoid passing null into methods

Do not pass null parameters into an API method unless the *API reference* states explicitly that the method supports them.

Use caution when passing null into a constructor

To avoid ambiguity when passing null into a constructor, cast null to the appropriate object.

```
new someObject ( (someObject)null );
```

If a class has two or more constructors, passing in a null parameter may not uniquely identify which constructor to use. As a result, the compiler reports an error. Not all supported constructors appear in the *API Reference* because some constructors are for internal use only.

By casting `null` to the appropriate object, you indicate precisely which constructor the compiler should use. This practice also provides forward compatibility if later releases of the API add new constructors.

Use longs for unique identifiers

Use a `long` identifier instead of a `String` identifier for unique constants, such as GUIDs, hash table keys, and state or context identifiers.

For identifiers to remain unique across third-party applications, use keys that are generated based on a hash of a `String`. In the input string, include enough information to provide uniqueness. For example, use a fully qualified package name such as `com.rim.samples.docs.helloworld`.

To convert a `String` to a `long`, complete the following actions:

1. In the IDE text editor, type a string.
2. Select the string.
3. Right-click, and then click **Convert "*string*" to Long**.

Exit applications correctly

Before you invoke `System.exit(int status)`, your application should perform any necessary cleanup, such as removing objects from the runtime store that are no longer required by any applications.

Print the stack trace

The VM is optimized to eliminate the stack trace if it finds code that catches the exception using `catch (Exception e)`. It does not eliminate the stack trace if `Throwable` is caught.

For example, the following code does not print a stack trace:

```
catch (IOException e) {
   e.printStackTrace()
}
```

To print a stack trace, write code such as the following:

```
catch (Throwable t) {
   t.printStackTrace();
}
```

Catch a `Throwable` instance only for debugging if you want to view the stack trace.

Creating user interfaces

User interface APIs

When you write applications for BlackBerry handhelds, use one of the following two API sets for user interfaces:

- MIDP UI APIs (`javax.microedition.lcdui` package)
- BlackBerry UI APIs (`net.rim.device.api.ui` packages)

If you are writing an application to run on any MIDP-compliant device, use the MIDP UI APIs. If you are writing an application specifically for BlackBerry handhelds, use the BlackBerry UI APIs. BlackBerry APIs provide access to specific features of the handheld and enable more sophisticated UI layout and interaction.

Note: Do not use MIDP UI APIs and BlackBerry UI APIs in the same application or exceptions will be thrown. The UI framework cannot support both types of UI objects in one application.

Displaying UI components

Displaying screens

The main structure for a user interface is the `Screen`. An application displays one `Screen` at a time.

Note: Do not use `Screen` objects for text input. The `Screen` class does not implement disambiguation, which is required for complex input methods such as international keyboards and the 7100 series of handhelds. For seamless integration of the different input methods, extend `Field` or one of its subclasses. See " Creating custom fields" on page 42 for more information.

The display stack

`Screen` objects are maintained in a display stack, an ordered set of `Screen` objects. The screen at the top of the stack is the active screen that appears to the user. When an application displays a screen, it pushes the screen to the top of the stack. When an application closes a screen, it pops the screen off the stack and displays the next screen on the stack, redrawing it as necessary.

Tip: Each screen can appear only once in the display stack. The VM throws a runtime exception if the same screen is pushed onto the stack more than once. Applications must pop screens off of the display stack when the user finishes interacting with them so that handheld memory is not used unnecessarily. Do not use more than a few modal screens at one time, because each screen uses a separate thread.

Types of screens

In most cases, the most efficient way to create a screen is to create a new class that extends `Screen` or one of its subclasses, `FullScreen` or `MainScreen`.

Class	Description
`Screen`	Use the `Screen` class to define a manager to lay out UI components on the screen.
`FullScreen`	By default, a `FullScreen` contains a single vertical field manager. Use a `FullScreen` to provide an empty screen that you can add UI components to in a standard vertical layout. If you need another type of layout, such as horizontal or diagonal, use a `Screen` class and add a `Manager` to it.
`MainScreen`	The `MainScreen` class provides features that are common to standard BlackBerry applications. Use a `MainScreen` object for the first screen of your application to maintain consistency with other BlackBerry applications. The `MainScreen` class provides the following UI components: • default position of a screen title, with a `SeparatorField` after the title • a main scrollable section contained in a `VerticalFieldManager` • default menu with a **Close** menu item • default close action when the user clicks the **Close** menu item or presses the **Escape** key

Listeners

The BlackBerry APIs provide an event listener framework that is similar to J2SE. In particular, two listener interfaces enable applications to receive and respond to user interaction: `TrackwheelListener` and `KeyboardListener`. The `Screen` class and its subclasses implement these interfaces.

Provide screen navigation

BlackBerry applications provide a menu for users to perform actions. Avoid using buttons or other UI elements that take up space on the screen. Users click the trackwheel to access the menu.

When you create a `FullScreen` or `Screen`, specify the `DEFAULT_MENU` and `DEFAULT_CLOSE` parameters in the constructor to provide default navigation.

```
FullScreen fullScreen = new FullScreen(DEFAULT_MENU | DEFAULT_CLOSE);
```

Parameter	Description
`DEFAULT_MENU`	This parameter adds a default menu, which includes different menu items, depending on the user context. For example, if users select an `EditField`, **Cut**, **Copy**, and **Paste** menu items appear. All selectable fields provide **Select** and **Cancel Selection** items.
`DEFAULT_CLOSE`	This parameter adds a **Close** item to the menu, with default behavior. When users click the **Close** menu item or press the **Escape** button, a confirmation dialog box appears if anything on the screen has changed. If this screen is the only one on the stack, the application closes.

Default navigation is provided by default when you create a `MainScreen`.

Add menu items

To add additional menu items, create `MenuItem` objects.

```
private MenuItem viewItem = new MenuItem("View Message", 100, 10) {
   public void run() {
      Dialog.inform("This is today's message");
   }
};
```

The `MenuItem` constructor accepts the following three parameters:

- name of the menu item

- order of menu items; a higher value indicates that the item appears closer to the bottom of the menu
- priority of the menu item for receiving the default focus

Implement `run()` to define the action that occurs when the user clicks the menu item. If you are not using localization resources, override `toString()` to specify the name of the menu item.

To add context menus to fields in an application, call `getLeafFieldWithFocus()` and invoke `getContextMenu()` on the return value to determine which fields receive custom menu items in `makeMenu()`. See " Creating custom context menus" on page 49 for more information.

When you add your own menu items, define a **Close** menu item explicitly:

```
private MenuItem closeItem = new MenuItem("Close", 200000, 10) {
public void run() {
        onClose();
    }
};
```

To add the menu items to the screen, override `Screen.makeMenu()`.

```
protected void makeMenu(Menu menu, int instance) {
    menu.add(viewItem);
    menu.add(closeItem);
}
```

If you extend `Screen` or one of its subclasses, the default implementation of `TrackwheelListener` invokes `makeMenu()` when the user clicks the trackwheel.

If you do not extend `Screen`, implement `TrackwheelListener`. In particular, implement `trackwheelClick()` to create a new `Menu`, add menu items, and display the menu on screen.

```
public boolean trackwheelClick(int status, int time) {
    Menu appMenu = new Menu();
    makeMenu(appMenu); // add menu items
    appMenu.show(); // display the menu on screen
    return true;
}
```

Tip: To create custom menu items for additional functionality, extend the `MenuItem` class. See " Creating custom context menus" on page 49 for more information.

Displaying dialog boxes

The `PopupScreen` class provides features for building dialog boxes and status screens using its subclasses, `Dialog` and `Status`. Popup screens are not pushed onto the display stack. To display a popup screen, invoke `Dialog.ask()` or `Status.show()`.

To control the layout of a dialog box, use a `DialogFieldManager` object. See " Specify the layout for a PopupScreen" on page 39 for more information.

Displaying dialog boxes

To display a dialog box, invoke `Dialog.ask()` with one of the following parameters:

Parameter	Description
D_OK	displays a string and prompts the user to click **OK**

Parameter	Description
D_SAVE	displays a string and prompts the user to click **Save**, **Discard**, or **Cancel**; pressing **Escape** returns cancel
D_DELETE	displays a string and prompts the user to click **Delete** or **Cancel**; pressing **Escape** returns cancel
D_YES_NO	displays a string and prompts the user to click **Yes** or **No**

```
int response = Dialog.ask(Dialog.D_SAVE);
if (Dialog.SAVE == response || Dialog.CANCEL == response)
   return false;
if ( Dialog.DISCARD == response )
   _item.deleteItem(_itemIndex);
```

To specify a default response for a dialog box, use a version of `Dialog.ask()` that accepts `defaultChoice` as a parameter.

```
int response = Dialog.ask(Dialog.D_YES_NO, "Are you sure?", Dialog.NO);
```

Displaying status messages

Invoke `Status.show()` to display a status message. By default, the status screen remains on the screen for two seconds.

```
Status.show("Status screen message");
```

See the *API Reference* for information on versions of `Status.show()` that enable you to specify additional parameters, such as a different icon or the amount of time that the status dialog remains visible. You can create status dialog boxes that are modal (require the user to dismiss them) or timed (dismiss automatically after a specified time).

Displaying fields

All UI components are represented by fields—rectangular regions that are contained in a manager. The size of the field is determined by its layout requirements. Managers provide scrolling for the fields that they contain.

The BlackBerry JDE provides a library of prebuilt interface controls and components in the `net.rim.device.api.ui.component` package. In most cases, you can use these objects to construct UI applications.

To create a specialized field component (such as a text field that contains multiple elements), create your own custom types by extending the `Field` class or one of its subclasses. See " Creating custom fields" on page 42 for more information.

Note: See the *API Reference* for more information on valid/supported styles for specific field classes. An exception is thrown if you instantiate a `Field` using an unsupported style.

Bitmap fields

A `BitmapField` contains bitmaps. Use a `BitmapField` when you draw with the `Graphics` object. To modify the contents of a field, invoke drawing methods on a `BitmapField`. See " Drawing using graphics objects" on page 62 for more information.

```
Bitmap myBitmap = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
BitmapField myBitmapField = new
   BitmapField(myBitmap.getPredefinedBitmap(myBitmap));
...
```

```
mainScreen.add(myBitmapField);
```

There are four predefined bitmaps:

- `Bitmap.INFORMATION`
- `Bitmap.QUESTION`
- `Bitmap.EXCLAMATION`
- `Bitmap.HOURGLASS`

To use an original .gif or .png image as a bitmap, invoke `getBitmapResource()`.

```
private static final Bitmap myBitmap =
   Bitmap.getBitmapResource("customBitmap.gif");
...
BitmapField bitmapField = new BitmapField(myBitmap);
mainScreen.add(bitmapField);
```

Button fields

A `ButtonField` contains buttons that users select to perform actions. Use `ButtonField` to create interfaces that have an extended interactivity beyond that of the menu.

```
ButtonField mySubmitButton = new ButtonField("Submit");
ButtonField myResetButton = new ButtonField("Reset");
mainScreen.add(mySubmitButton);
mainScreen.add(myResetButton);
```

To add functionality to the button, extend `ButtonField` and override `trackwheelClick()` so that it performs an action instead of invoking the menu. To receive notification when the user clicks the button, use a `FieldChangeListener` object. See " Listening for changes to UI objects" on page 67 for more information.

Choice fields

Choice fields are similar to drop-down lists. There are two types of choice fields: those that contain integers and those that contain objects that can be converted to strings.

You can also display a set of options as check boxes or radio buttons. See " Option fields" on page 34 for more information.

To select a value from the `ChoiceField`, users perform one of the following actions:

- click the field and press the **Space** key
- hold the **Alt** key and roll the trackwheel
- open the menu and click **Change Option**

NumericChoiceField

A `NumericChoiceField` is a `ChoiceField` that contains a range of numeric values. `NumericChoiceField` instances are typically used for a small range of numbers (up to 20).

```
NumericChoiceField myNumericChoice = new NumericChoiceField( "Select a number: ",
   1, 20, 10);
mainScreen.add(myNumericChoice);
```

Tip: For a large range of numbers, use a `GaugeField`. See " Gauge fields" on page 36 for more information.

ObjectChoiceField

An `ObjectChoiceField` is a `ChoiceField` that contains objects. All objects in the field should implement `Object.toString()` to provide string representations of themselves.

Provide an object array as a parameter when you create an `ObjectChoiceField`.

```
String choiceItems[] = {"Option one", "Option two", "Option three"};
mainScreen.add(new ObjectChoiceField("Select an option:", choiceItems));
```

A **Change Option** menu item is provided by default for an `ObjectChoiceField`. Users click **Change Option** and select an option.

Option fields

Option fields enable users to select entries from lists. Use `CheckboxField` for option lists that enable users to select multiple entries. Use a `RadioButtonField` for lists that enable users to select only one entry.

CheckboxField

Each `CheckboxField` object is an individual object that is not associated with the other check boxes.

```
CheckboxField myCheckbox = new CheckboxField("First checkbox", true);
CheckboxField myCheckbox2 = new CheckboxField("Second checkbox", false);
...
mainScreen.add(myCheckbox);
mainScreen.add(myCheckbox2);
```

RadioButtonField

Multiple `RadioButtonField` objects are combined into a `RadioButtonGroup` so that the user can select only one option at a time.

```
RadioButtonGroup rbGroup = new RadioButtonGroup();
RadioButtonField rbField = new RadioButtonField("First field");
RadioButtonField rbField2 = new RadioButtonField("Second field");
...
rbGroup.add(rbField);
rbGroup.add(rbField2);
...
mainScreen.add(rbField);
mainScreen.add(rbField2);
```

Date fields

A `DateField` displays the current date and time in your application.

Type	Description
DATE	displays the year, month, and day
DATE_TIME	displays the year, month, day, hour, minutes, and seconds
TIME	displays the hour, minutes and seconds

When you create a `DateField`, call `System.currentTimeMillis()` to retrieve the current time.

```
DateField dateField = new DateField("Date: ", System.currentTimeMillis(),
   DateField.DATE_TIME);
mainScreen.add(dateField);
```

Date fields are editable by default. To create a `DateField` that cannot be edited by users, specify the `Field.READONLY` parameter in the `DateField` constructor.

A **Change Option** menu item is provided by default for an editable `DateField`.

Edit fields

An `EditField` enables users to type text in fields. `AutoTextEditField`, `EditField`, and `PasswordEditField` extend `BasicEditField`.

Note: The `net.rim.device.api.ui.component.TextField` class, which extends the `Field` class, is abstract. Instantiate one of its subclasses, such as `RichTextField` or `EditField`, to create a UI field that displays text or enables a user to type text.

You can apply the following filters to edit fields:

Filter	Description
DEFAULT_MAXCHARS	This filter limits the number of characters in the field. The default maximum number of characters for edit fields is 15.
FILTER_DEFAULT	This is the default text input filter. Use this filter when the constructor requires a filter but you do not want to apply any special filtering.
FILTER_EMAIL	This filter permits only valid email address characters (for example, users can only type one @ sign).It automatically formats text into email address format (for example, when the user presses the space key for the first time, an @ symbol appears, followed by .'s each additional time the user presses the space key).
FILTER_HEXADECIMAL	This filter permits only numbers and the letters A through F.
FILTER_INTEGER	This filter permits only numbers and the minus sign (–).
FILTER_LOWERCASE	This filter converts letters to lowercase.
FILTER_NUMERIC	This filter permits only numbers.
FILTER_PHONE	This filter permits only valid phone number characters, numeric characters, hyphen, plus and minus signs, right and left parentheses, and " x" .
FILTER_PIN_ADDRESS	This filter accepts only characters valid in a PIN address for input.
FILTER_UPPERCASE	This filter converts letters to uppercase.
FILTER_URL	This filter permits only valid URL characters. It also automatically formats fields (for example, it inserts a period when the user presses the space key).
JUMP_FOCUS_AT_END	This filter changes field behavior, so that when the field is in focus, and the user attempts to scroll down, the focus moves to the end of the field (instead of moving to the next field).
NO_NEWLINE	This filter ignores line feeds and carriage returns in text, such as text that a user copies and pastes from another source.

RichTextField

`RichTextField` creates a read-only field that can be formatted with different fonts and styles. Rich text fields, although not editable, can receive focus.

```
mainScreen.add(new RichTextField("RichTextField"));
```

BasicEditField

`BasicEditField` is the base class for `EditField` and `PasswordEditField`.

`BasicEditField` is an editable text field that contains no default formatting, but accepts filters.

```
BasicEditField bf = new BasicEditField("BasicEditField: ", "", 10,
   EditField.FILTER_UPPERCASE);
mainScreen.add(bf);
```

EditField

EditField is an editable text field that extends BasicEditField. EditField enables users to access special characters. For example, users hold the A key and roll the trackwheel to select from a variety of accented A characters, and the Æ ligature. EditField accepts styles, but some styles negate the functionality of EditField (such as EditField.FILTER_PHONE).

```
mainScreen.add(new EditField("EditField: ", "", 10, EditField.FILTER_DEFAULT));
```

PasswordEditField

PasswordEditField extends BasicEditField. It masks user input with asterisk characters (*). AutoText (and other automatic formatting) is not applied and cut or copy operations are not supported.

The following example uses a constructor that enables you to provide a default initial value for the PasswordEditField.

```
mainScreen.add(new PasswordEditField("PasswordEditField: ", ""));
```

AutoTextEditField

AutoTextEditField applies formatting that is specified by the AutoText engine. Any text that is typed in this field is formatted according to the specifications of the AutoText database on the handheld.

Some filters render some AutoText entries ineffective. For example, FILTER_LOWERCASE renders an AutoText entry that contains capitalization ineffective.

```
mainScreen.add(new AutoTextEditField("AutoTextEditField: ", ""));
```

Gauge fields

Gauges enable you to create visual representations of numeric values. GaugeField displays a progress bar, or enables users to select numbers. You can prefix the gauge with a label, and display the current value within the gauge. For example, combine a GaugeField and a NumericChoiceField to create a graphical representation of a numeric selection made by the user.

To create an interactive GaugeField, instantiate the field with Field.FOCUSABLE and Field.EDITABLE styles.

```
GaugeField staticGauge = new GaugeField("1: ", 1, 100, 20, GaugeField.NO_TEXT);
GaugeField percentGauge = new GaugeField("Percent: ", 1, 100, 29,
   GaugeField.PERCENT)
GaugeField interactiveGauge = new GaugeField("Gauge: ", 1, 100, 60, Field.FOCUSABLE
   | Field.EDITABLE);
...
mainScreen.add(staticGauge);
mainScreen.add(percentGauge);
mainScreen.add(interactiveGauge);
```

Label and separator fields

A LabelField enables you to add text labels to screens. A LabelField is read-only. By default, it cannot receive focus. Most applications use a LabelField to display a static title on their first screen.

A SeparatorField is a static horizontal line that spans the width of the screen. Use a SeparatorField to group related content on screens and menus.

The MainScreen displays a separator after the title by default.

```
LabelField title = new LabelField("UI Component Sample", LabelField.ELLIPSIS));
mainScreen.setTitle(title);
```

List fields

Lists enable you to create directories of items through which users can scroll and select individual or multiple entries. There are three types of list objects: `ListField`, `ObjectListField`, and `TreeField` objects. The BlackBerry address book is an example of a `List` object.

You cannot directly populate the field entries with content. To draw the field, implement a `ListFieldCallback` for a `ListField` and a `TreeFieldCallback` for a `TreeField`.

ListField

`ListField` contains rows of selectable items. To display content in a `ListField`, set a `ListFieldCallback` for the list. See " Implement the ListFieldCallback interface" on page 56 for more information.

```
String fieldOne = new String("Mark Guo");
String fieldTwo = new String("Amy Krul");
...
ListField myList = new ListField();
ListCallback myCallback = new ListCallback();
myList.setCallback(myCallback);
myCallback.add(myList, fieldOne);
myCallback.add(myList, fieldTwo);
...
mainScreen.add(myList);
```

Tip: To enable the user to select a range of items in the list, specify a `ListField` as `MULTI_SELECT`.

`ListFieldCallback.add()` adds the list element to the vector and calls `List.insert()` to determine the appropriate position.

ObjectListField

An `ObjectListField` is a list field that contains objects as entries. All objects that are contained in the list must implement `Object.toString()` to provide string representations of themselves. An `ObjectListField` is rendered in the interface in the same way as a standard `ListField`.

TreeField

`TreeField` contains parent and child nodes and presents a folder or tree relationship between items (such as documents or message folders). By default, all the entries are visible. To specify whether a folder is collapsible, invoke `setExpanded()` on the `TreeField` object.

Icons appear beside each node that has child nodes to specify whether the node is expanded or collapsed.

```
String fieldOne =  new String("Main folder");
...
TreeCallback myCallback = new TreeCallback();
TreeField myTree = new TreeField(myCallback, Field.FOCUSABLE);
int node1 = myTree.addChildNode(0, fieldOne);
int node2 = myTree.addChildNode(0, fieldTwo);
int node3 = myTree.addChildNode(node2, fieldThree);
int node4 = myTree.addChildNode(node3, fieldFour);
...
int node10 = myTree.addChildNode(node1, fieldTen);
myTree.setExpanded(node4, false);
...
mainScreen.add(myTree);
```

Implement a `TreeFieldCallback` to add fields to the tree. See " Implement the ListFieldCallback interface" on page 56 for more information on callbacks.

```
private class TreeCallback implements TreeFieldCallback {
   public void drawTreeItem(TreeField _tree, Graphics g, int node, int y, int
      width, int indent) {
      String text = (String)_tree.getCookie(node);
      g.drawText(text, indent, y);
   }
}
```

Managing UI components

Managing layout

Use BlackBerry API layout managers to arrange components on a screen.

The following four classes extend the `Manager` class to provide predefined layout managers:

- `VerticalFieldManager`
- `HorizontalFieldManager`
- `FlowFieldManager`
- `DialogFieldManager`

Both `MainScreen` and `FullScreen` use a `VerticalFieldManager` by default; define a layout manager for instances of these classes only to provide a different layout.

Tip: To create a custom layout manager, extend `Manager`. See " Creating custom layout managers" on page 52 for more information.

To define the layout manager for a particular instance of `Screen`, complete the following actions:

- instantiate the appropriate `Manager` subclass
- add UI components to the layout manager
- add the layout manager to the screen

```
VerticalFieldManager vfm = new VerticalFieldManager(Manager.VERTICAL_SCROLL);
vfManager.add(bitmapField);
vfManager.add(bitmapField2);
...
mainScreen.add(vfManager)
```

The `Manager` class defines several constants that provide system styles, which define behavior such as scrolling and alignment. Use these styles as parameters when you create the layout manager. See the *API reference* for more information.

Organize fields vertically

`VerticalFieldManager` organizes fields vertically. All fields start on a new line. To enable vertical scrolling, provide the `Manager.VERTICAL_SCROLL` parameter.

```
VerticalFieldManager vfm = new VerticalFieldManager(Manager.VERTICAL_SCROLL);
vfManager.add(bitmapField);
vfManager.add(bitmapField2);
```

```
...
mainScreen.add(vfManager);
```

By default, `BitmapField` objects are all left-aligned in the `VerticalFieldManager`.

Organize fields horizontally

`HorizontalFieldManager` organizes fields horizontally. To enable horizontal scrolling, provide the `Manager.HORIZONTAL_SCROLL` style. If you do not include the `HORIZONTAL_SCROLL` parameter, the fields arrange themselves horizontally and can exceed the width of the screen, but users cannot scroll to content that is beyond the right side of the screen.

Handhelds do not display horizontal scrolling indicators or scroll bars.

```
HorizontalFieldManager hfm = new
   HorizontalFieldManager(Manager.HORIZONTAL_SCROLL);
```

Organize fields horizontally and vertically

`FlowFieldManager` organizes fields horizontally, and then vertically. Fields are arranged horizontally until there is insufficient space to place another field, and then the manager arranges them horizontally on the next line. The Home screen is an example of a `FlowFieldManager`.

```
FlowFieldManager flManager = new FlowFieldManager(Manager.FIELD_HCENTER);
```

Specify the layout for a PopupScreen

`DialogFieldManager` specifies the layout for `PopupScreen` objects. It manages layout for an icon, a message, and a list of custom fields. The icon and message appear beside each other at the top of the layout, and the custom fields appear below the message. This layout is standard for dialog `PopupScreen` objects. Use `DialogFieldManager` as a base class for creating your own dialog boxes.

```
BitmapField bitmapField = new BitmapField(Bitmap.getBitmapResource("x.gif"));
RichTextField message = new RichTextField("Dialog manager message",
   Field.NON_FOCUSABLE);
LabelField dialogChoice = new LabelField("Choice one", Field.FOCUSABLE);
...
DialogFieldManager dialogManager = new DialogFieldManager();
dialogManager.setMessage(message);
dialogManager.setIcon(bitmapField);
dialogManager.addCustomField(dialogChoice);
```

Managing UI interactions

Only one thread at a time (usually the event-dispatching thread) can gain access to the UI. Background threads can access the UI from outside the main event-handling or UI drawing code in one of the following ways:

- acquire and hold the event lock
- use `invokeLater()` or `invokeAndWait()` to run on the event dispatch thread

Acquire and hold the event lock

The event dispatcher sets an event lock on the event thread while it processes a message. Background threads (that is, non-event dispatch threads) can access the UI by acquiring this lock for a short time, without interfering with event dispatcher processing.

To retrieve the event lock, invoke `Application.getEventLock()`. Synchronize with this object to provide serialized access to the UI. Hold this lock for only short periods of time because the lock pauses the event dispatcher. An application should never call `notify()` or `wait()` on the `EventLock` object.

```
class MyTimerTask extends TimerTask {
   public void run() {
      synchronized(Application.getEventLock()) {
         _label.setText("new text " + System.currentTimeMillis());
      }
   }
}
```

Run on the event dispatch thread

If holding the event lock is not appropriate, create a class that implements the `Runnable` interface. Invoke its `run()` on the event dispatch thread by invoking one of the following three methods:

- Invoke `invokeAndWait(Runnable runnable)` to have `run()` called on the event dispatch thread immediately. The call blocks until `run()` completes.
- Invoke `invokeLater(Runnable runnable)` to have `run()` called on the event dispatch thread after all pending events are processed.
- Invoke `invokeLater(Runnable runnable, long time, boolean repeat)` to have `run()` called on the event dispatch thread after a specified amount of time, where `time` specifies the number of milliseconds to wait before adding `Runnable` to the event queue. If `repeat` is `true`, the `Runnable` is added to the event queue every `time` milliseconds.

Managing foreground events

The system calls `Application.activate()` when it brings an application to the foreground.

Most applications do not need to override `activate()`. Applications should perform any initialization, including any required `UiApplication.pushScreen()` calls, in the application constructor. Because `activate()` can be called multiple times for the same application, applications should not perform a one-time initialization in this method.

An application can override `activate()` to perform additional processing when it is brought to the foreground. If you override `activate()`, invoke `super.activate()` from within the method definition so that the application repaints correctly.

Managing drawing areas

Using XYRect objects

The `Graphics` object represents the entire drawing surface that is available to the application. To limit this area, divide it into `XYRect` objects. An `XYRect` creates rectangular clipping regions on top of the graphics context.

An `XYRect` object consists of two `XYPoint` objects. The first `XYPoint` object represents the top left coordinate of the `XYRect` and the second `XYPoint` represents the bottom right coordinate. Each `XYPoint` represents a point on the screen, which is composed of an X coordinate and a Y coordinate.

```
XYPoint topLeft = new XYPoint(10, 10);
XYPoint bottomRight = new XYPoint(50, 50);
XYRect rectangle = new XYRect(topLeft, bottomRight);
```

The `rectangle` object limits the drawing area of the context for this `XYRect` object to the area between `(10, 10)` and `(50, 50)`.

To initiate drawing calls to the XYRect object, invoke `pushContext()` or `pushRegion()`.

When you make drawing calls with `pushContext()`, specify that the region origin should not adjust the drawing offset.

```
graphics.pushContext(rectangle, 0, 0);
graphics.fillRect(10, 10, 30, 30);
graphics.drawRect(15, 15, 30, 30);
graphics.popContext();
```

When you invoke drawing methods by first calling `pushRegion()`, you specify that the drawing offset is to be adjusted by the region origin. The top left `XYPoint` object represents the region origin. All drawing is offset by this amount.

In the following example, `pushRegion()` places the `XYRect` object 10 pixels to the right of, and ten pixels down, from `(10, 10)`. The region origin adjusts the drawing offset (`XYPoint topLeft = new XYPoint(10, 10)`).

```
graphics.pushRegion(rectangle);
graphics.fillRect(10, 10, 30, 30);
graphics.drawRect(15, 15, 30, 30);
graphics.popRegion();
```

Invert an area

Inverting an area on the `Graphics` object reverses the pixels by inverting the bits in each pixel value (that is, 0s become 1s, 1s become 0s). Most fields use inversion to signify focus; however, you can create your own focus behavior for custom fields.

To invert an arbitrary portion of the `Graphics` object, provide coordinates, or invert a specified `XYRect` object.

Specify the portion of the `Graphics` object to push onto the stack. After you invoke `pushContext()` (or `pushRegion()`), provide the portion of the `Graphics` object to invert.

```
graphics.pushContext(rectangle);
graphics.invert(rectangle); // invert the entire XYRect object
graphics.popContext();
```

Translate an area

To move an area to another location in the graphics context, invoke `translate()`.

```
XYRect rectangle = new XYRect(1, 1, 100, 100);
XYPoint newLocation = new XYPoint(20, 20);
rectangle.translate(newLocation);
```

The `XYRect` is translated from its origin of `(1, 1)` to an origin of `(20, 20)`. After translation, the bottom portion of the `XYRect` object extends past the bounds of the graphics context and is clipped.

Creating custom UI components

You can only add custom context menu items and custom layouts to a custom field.

Creating custom fields

To override default field behaviour, create a custom field.

Note: Do not use Screen objects for text input. Screen does not implement disambiguation, which is required for complex input methods such as international keyboards. For seamless integration of the different input methods, extend Field or one of its subclasses.

To enable drawing styles on custom fields, implement the DrawStyle interface. See " Implementing the DrawStyle interface" on page 63 for more information.

Your custom field should implement all relevant system styles. For example, USE_ALL_WIDTH and USE_ALL_HEIGHT are appropriate for many fields.

Extend the Field class

Extend the Field class, or one of its subclasses, to specify the characteristics of the custom field.

```
public class CustomButtonField extends Field implements DrawStyle {
   public static final int RECTANGLE = 1;
   public static final int TRIANGLE = 2;
   public static final int OCTAGON = 3;
   private String _label;
   private int _shape;
   private Font _font;
   private int _labelHeight;
   private int _labelWidth;
}
```

Implement constructors

To define the button label, shape, and style, implement constructors.

```
public CustomButtonField(String label) {
   this(label, RECTANGLE, 0);
}
public CustomButtonField(String label, int shape) {
   this(label, shape, 0);
}
public CustomButtonField(String label, long style) {
   this(label, RECTANGLE, style);
}
public CustomButtonField(String label, int shape, long style) {
   super(style);
   _label = label;
   _shape = shape;
   _font = getFont();
   _labelHeight = _font.getHeight();
   _labelWidth = font.getWidth();
}
```

Implement layout()

Any class that extends Field must implement layout(). The field manager invokes layout() to determine how the field should arrange its contents, according to the available space.

```
protected void layout(int width, int height) {
   _font = getFont();
```

```
    _labelHeight = _font.getHeight();
    _labelWidth = _font.getAdvance(_label);
    width = Math.min( width, getPreferredWidth() );
    height = Math.min( height, getPreferredHeight() );
    setExtent( width, height );
}
```

The `width` and `height` parameters specify the available horizontal and vertical space, respectively. To calculate the available width and height, invoke `Math.min()` to return the smaller of the specified width and height and the preferred width and height of the field. See " Return a preferred field size" on page 43 for more information.

To set the required dimensions for the field, invoke `setExtent(int, int)`. If you do not invoke `setExtent()`, the field is not painted and an exception is not thrown.

Tip: Arrange field data so that you perform the most complex calculations in `layout()`, instead of in `paint()`. Implement `paint()` to be as efficient as possible.

Recalculate pixel layout, cached fonts, and locale strings in `layout()`. The framework invokes this method whenever system preferences change. For example, when the system default font changes, all fields in the system update automatically if their layout method is implemented correctly. The same is true for locale changes and date format changes.

Return a preferred field size

In most cases, override `getPreferredWidth()` and `getPreferredHeight()` to make sure that the proper layout appears in custom layout managers.

Define the preferred width

Implement `getPreferredWidth()` the preferred width of the custom button based on the relative dimensions of the button label. Using the relative dimensions makes sure that the label does not exceed the button dimensions.

```
public int getPreferredWidth() {
    switch(_shape) {
        case TRIANGLE:
            if (_labelWidth < _labelHeight) {
                return _labelHeight << 2;
            } else {
                return _labelWidth << 1;
            }
        case OCTAGON:
            if (_labelWidth < _labelHeight) {
                return _labelHeight + 4;
            } else {
                return _labelWidth + 8;
            }
        case RECTANGLE: default:
            return _labelWidth + 8;
    }
}
```

Define the preferred height

Implement `getPreferredHeight()` to determine the preferred height of the custom button based on the relative dimensions of the button label. This ensures that the label does not exceed the button dimensions.

```
public int getPreferredHeight() {
    switch(_shape) {
            case TRIANGLE:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight << 1;
                } else {
                    return _labelWidth;
                }
            case RECTANGLE:
                return _labelHeight + 4;
            case OCTAGON:
                return getPreferredWidth();
        }
        return 0;
    }
```

Define the appearance of the custom field

To define the appearance of the custom field on handheld screens, implement `paint()`. The field manager invokes `paint()` to redraw the field whenever an area of the field is marked as invalid.

Tips: Verify that `paint()` is efficient because the UI framework calls `paint()` whenever an area of the field changes. For large fields, use `Graphics.getClippingRect()` to save drawing time by painting only within the visible region. Avoid allocating in `paint()`; arrange field data so that complex calculations are performed in `layout()` instead of in `paint()`.

```
protected void paint(Graphics graphics) {
    int textX, textY, textWidth;
    int w = getWidth();
    switch(_shape) {
        case TRIANGLE:
            int h = (w>>1);
            int m = (w>>1)-1;
            graphics.drawLine(0, h-1, m, 0);
            graphics.drawLine(m, 0, w-1, h-1);
            graphics.drawLine(0, h-1, w-1, h-1);
            textWidth = Math.min(_labelWidth,h);
            textX = (w - textWidth) >> 1;
            textY = h >> 1;
            break;
        case OCTAGON:
            int x = 5*w/17;
            int x2 = w-x-1;
            int x3 = w-1;
            graphics.drawLine(0, x, 0, x2);
            graphics.drawLine(x3, x, x3, x2);
            graphics.drawLine(x, 0, x2, 0);
            graphics.drawLine(x, x3, x2, x3);
            graphics.drawLine(0, x, x, 0);
            graphics.drawLine(0, x2, x, x3);
            graphics.drawLine(x2, 0, x3, x);
            graphics.drawLine(x2, x3, x3, x2);
            textWidth = Math.min(_labelWidth, w - 6);
            textX = (w-textWidth) >> 1;
            textY = (w-_labelHeight) >> 1;
            break;
        case RECTANGLE: default:
            graphics.drawRect(0, 0, w, getHeight());
            textX = 4;
            textY = 2;
            textWidth = w - 6;
            break;
    }
    graphics.drawText(_label, textX, textY, (int)( getStyle() & DrawStyle.ELLIPSIS
        | DrawStyle.HALIGN_MASK ), textWidth );
}
```

Handle focus events

To support focus events, use the `Field.FOCUSABLE` style and implement `Field.moveFocus()`. If you want your field to receive focus, override `Field.isFocusable()` to return `true`.

The framework invokes `onFocus()` when the field gains focus and `onUnfocus()` when the field loses focus. Override these methods if your field requires specific behavior for these events. The framework invokes `moveFocus()` to handle focus movements in a field. This corresponds to the `trackwheelRoll` event. Override `moveFocus()` for special behavior.

To change the appearance of the default focus indicator (inverting the contents of the entire field), override `drawFocus()`.

Implement set and get methods

To add capabilities to your field, implement the various get and set methods of `Field`.

Note: All accessor and mutator (get and set) methods should work before and after the field is added to a `Screen`. For example, `setLabel()` should update the display with the new value if the field is currently on screen by invoking `invalidate()` or `updateLayout()` as appropriate.

```
public String getLabel() {
    return _label;
}
public int getShape() {
    return _shape;
}
public void setLabel(String label) {
    _label = label;
    _labelWidth = _font.getAdvance(_label);
    updateLayout();
}
public void setShape(int shape) {
    _shape = shape;
    updateLayout();
}
```

Code example

The CustomButtonField.java sample creates button fields of various shapes.

Example: CustomButtonField.java

```
/**
 * CustomButtonField.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.custombuttons;
```

```
import net.rim.device.api.ui.*;
import net.rim.device.api.system.*;

/**
 * CustomButtonField  is a class that creates button fields of various
 * shapes. Demonstrates how to create custom UI fields.
 */

public class CustomButtonField extends Field implements DrawStyle {
    public static final int RECTANGLE = 1;
    public static final int TRIANGLE = 2;
    public static final int OCTAGON = 3;

    private String _label;
    private int _shape;
    private Font _font;
    private int _labelHeight;
    private int _labelWidth;

    /* Constructs a button with specified label, and the default style and shape.
   */
    public CustomButtonField(String label) {
        this(label, RECTANGLE, 0);
    }

    /* Constructs a button with specified label and shape, and the default style.
   */
    public CustomButtonField(String label, int shape) {
        this(label, shape, 0);
    }

    /* Constructs a button with specified label and style, and the default shape.
   */
    public CustomButtonField(String label, long style) {
        this(label, RECTANGLE, style);
    }
    /* Constructs a button with specified label, shape, and style */
    public CustomButtonField(String label, int shape, long style) {
        super(style);
        _label = label;
        _shape = shape;
        _font = getFont();
        _labelHeight = _font.getHeight();
        _labelWidth = _font.getAdvance(_label);
    }

    /* Method that draws the focus indicator for this button and
     * inverts the inside region of the shape.
     */
    protected void drawFocus(Graphics graphics, boolean on) {
        switch(_shape) {
            case TRIANGLE:
                int w = getWidth();
                int h = w >> 1;
                for (int i=h-1; i>=2; --i) {
```

```
                graphics.invert(i, h - i, w - (i << 1), 1);
            }
            break;
        case RECTANGLE:
            graphics.invert(1, 1, getWidth() - 2, getHeight() - 2);
            break;
        case OCTAGON:
            int x3 = getWidth();
            int x = 5 * x3 / 17;
            int x2 = x3 - x;
            x3 = x3 - 1;
            x2 = x2 - 1;
            graphics.invert(1, x, getWidth() - 2, x2 - x + 1);

            for (int i=1; i<x; ++i) {
                graphics.invert(1+i, x-i,
                        getWidth() - ((i+1)<<1), 1);
                graphics.invert(1+i, x2+i,
                        getWidth() - ((i+1)<<1), 1);
            }
            break;
        }
}

/* Returns the label. */
public String getLabel() {
    return _label;
}
/* Returns the shape. */
public int getShape() {
    return _shape;
}

/* Sets the label. */
public void setLabel(String label) {
    _label = label;
    _labelWidth = _font.getAdvance(_label);

    updateLayout();
}

/* Sets the shape. */
public void setShape(int shape) {
    _shape = shape;
    updateLayout();
}
/* Retrieves the preferred width of the button. */
public int getPreferredWidth() {
    switch(_shape) {
        case TRIANGLE:
            if (_labelWidth < _labelHeight) {
                return _labelHeight << 2;
            } else {
                return _labelWidth << 1;
            }
            case OCTAGON:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight + 4;
                } else {
                    return _labelWidth + 8;
                }
            case RECTANGLE: default:
                return _labelWidth + 8;
        }
    }

    /* Retrieves the preferred height of the button. */
    public int getPreferredHeight() {
        switch(_shape) {
            case TRIANGLE:
                if (_labelWidth < _labelHeight) {
                    return _labelHeight << 1;
                } else {
                    return _labelWidth;
                }
            case RECTANGLE:
                return _labelHeight + 4;
            case OCTAGON:
                return getPreferredWidth();
        }
        return 0;
    }
    /**
     * Lays out this button’s contents.
     * This field’s manager invokes this method during the layout
     * process to instruct this field to arrange its contents, given an
     * amount of available space.
     **/
    protected void layout(int width, int height) {
        // Update the cached font in case it has been changed.
        _font = getFont();
        _labelHeight = _font.getHeight();
        _labelWidth = _font.getAdvance(_label);

        // Calculate width.
        width = Math.min( width, getPreferredWidth() );
        // Calculate height.
        height = Math.min( height, getPreferredHeight() );
        // Set dimensions.
        setExtent( width, height );
    }

    /*
     * Redraws this button. The field’s manager invokes this method during the
     * repainting process to instruct this field to repaint itself
     */
    protected void paint(Graphics graphics) {
        int textX, textY, textWidth;
        int w = getWidth();
        switch(_shape) {
                case TRIANGLE:
                    int h = (w>>1);
                    int m = (w>>1)-1;
                    graphics.drawLine(0, h-1, m, 0);
                    graphics.drawLine(m, 0, w-1, h-1);
                    graphics.drawLine(0, h-1, w-1, h-1);

                    textWidth = Math.min(_labelWidth,h);
                    textX = (w - textWidth) >> 1;
                    textY = h >> 1;
                    break;
                case OCTAGON:
                    int x = 5*w/17;
                    int x2 = w-x-1;
                    int x3 = w-1;
                    graphics.drawLine(0, x, 0, x2);
                    graphics.drawLine(x3, x, x3, x2);
                    graphics.drawLine(x, 0, x2, 0);
                    graphics.drawLine(x, x3, x2, x3);
                    graphics.drawLine(0, x, x, 0);
                    graphics.drawLine(0, x2, x, x3);
                    graphics.drawLine(x2, 0, x3, x);
                    graphics.drawLine(x2, x3, x3, x2);
                    textWidth = Math.min(_labelWidth, w - 6);
                    textX = (w-textWidth) >> 1;
                    textY = (w-_labelHeight) >> 1;
                    break;

                case RECTANGLE: default:
                    graphics.drawRect(0, 0, w, getHeight());
                    textX = 4;
                    textY = 2;
                    textWidth = w - 6;
                    break;
            }
            graphics.drawText(_label, textX, textY, (int)( getStyle() &
                DrawStyle.ELLIPSIS | DrawStyle.HALIGN_MASK ),
                textWidth );
        }
}
```

Creating custom context menus

The system invokes `getContextMenu()` when context menu events occur. It invokes `makeContextMenu()` to reset the context menu with the menu items that are appropriate for that field. The context menu contents remain set until the next time `getContextMenu()` is invoked.

To add custom context menu items to specific fields, create custom context menu items and then override `makeContextMenu()` and `makeMenu()` to provide and display a context menu.

Custom context menus can only be added to custom fields. See " Creating custom fields" on page 42 for more information.

Create custom context menu items

In your field class, create the custom context menu items. See " Displaying screens" on page 29 for more information on implementing menus.

```
private MenuItem myContextMenuItemA = new MenuItem( _resources, MENUITEM_ONE,
   200000, 10) {
   public void run() {
      onMyMenuItemA();
   }
};
private MenuItem myContextMenuItemB = new MenuItem( _resources, MENUITEM_ONE,
   200000, 10) {
   public void run() {
      onMyMenuItemB();
   }
};
```

Provide a context menu

In your main application class, override `makeContextMenu()` to provide a context menu.

```
protected void makeContextMenu(ContextMenu contextMenu) {
   contextMenu.addItem(myContextMenuItemA);
   contextMenu.addItem(myContextMenuItemB);
}
```

Create the application menu

In your main application class, override `makeMenu()` to create the application menu and update the context menu whenever a particular field has the focus.

```
protected void makeMenu(Menu menu) {
   Field focus =
      UiApplication.getUiApplication().getActiveScreen().getLeafFieldWithFocus();
   if (focus != null) {
      ContextMenu contextMenu = focus.getContextMenu();
      if (!contextMenu.isEmpty()) {
         menu.add(contextMenu);
         menu.addSeparator();
      }
   }
}
```

Code example

Example: ContextMenuSample.java

```
/**
 * ContextMenuSample.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.contextmenus;

import net.rim.device.api.i18n.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import com.rim.samples.docs.baseapp.*;

public class ContextMenuSample extends BaseApp implements
   ContextMenuSampleResource {

    private MyContextField myContextField;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
                ContextMenuSampleResource.BUNDLE_ID,
                ContextMenuSampleResource.BUNDLE_NAME);

    public static void main(String[] args) {
        ContextMenuSample app = new ContextMenuSample();
        app.enterEventDispatcher();
    }

    // Inner class to define a new field.
    private class MyContextField extends RichTextField {
        private MenuItem myContextMenuItemA = new MenuItem(
                _resources, MENUITEM_ONE, 200000, 10) {
            public void run() {
                onMyMenuItemA();
            }
        };
        private MenuItem myContextMenuItemB = new MenuItem(
            _resources, MENUITEM_TWO, 200000, 10) {
                public void run() {
                    onMyMenuItemB();
                }
            };

        private void onMyMenuItemA() {
            // Perform an action when user selects menu item.
        }
        private void onMyMenuItemB() {
            // Perform an action when user selects menu item.
        }

        protected void makeContextMenu(ContextMenu contextMenu) {
            contextMenu.addItem(myContextMenuItemA);
            contextMenu.addItem(myContextMenuItemB);
        }

        MyContextField(String text) {
            super(text);
        }
    }

    protected void makeMenu(Menu menu) {
        super.makeMenu(menu, 0); // Implemented by BaseApp.
    }

    public ContextMenuSample() {
        MainScreen mainScreen = new MainScreen();

        MyContextField myContextField = new MyContextField("Field label: ");
        mainScreen.add(myContextField);

        mainScreen.addKeyListener(this);
        mainScreen.addTrackwheelListener(this);

        pushScreen(mainScreen);

    }

    public void onExit() {
        // Perform action when application closes.
    }
}
```

Creating custom layout managers

`Manager` objects control the position of UI components and determine how fields are organized on the handheld screen.

To create a custom layout manager, extend the `Manager` class and customize it as follows:

- Implement `getPreferredWidth()` and `getPreferredHeight()`.
- Implement `sublayout()`.
- Handle focus.
- Implement `subpaint()`.

Extend Manager

To create a custom layout manager, extend the `Manager` class or one of its subclasses.

```
class DiagonalManager extends Manager {
   public DiagonalManager(long style){
      super(style);
   }
   ...
}
```

Implement getPreferredWidth()

Override `getPreferredWidth()` so that it returns the preferred field width for the manager.

Implementations of `getPreferredWidth()` return a different value depending on the purpose of the layout manager. For example, if a manager extends `HoriztonalFieldManager`, `getPreferredWidth()` returns the sum of the widths of each field. If a manager extends `VerticalFieldManager`, `getPreferredWidth()` returns the width of the widest field.

```
public int getPreferredWidth() {
   int width = 0;
   int numberOfFields = getFieldCount();
   for (int i=0; i<numberOfFields; ++i) {
      width += getField(i).getPreferredWidth();
```

```
    }
    return width;
}
```

Note: TextFields and Managers use the entire width that is assigned to them. To organize two or more of these objects horizontally, override their respective getPreferredWidth() methods accordingly. To organize multiple TextFields horizontally, override layout().

Implement getPreferredHeight()

Override getPreferredHeight() so that it returns the preferred field height for the manager.

```
public int getPreferredHeight() {
    int height = 0;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
        height += getField(i).getPreferredHeight();
    }
    return height;
}
```

Implement sublayout()

The sublayout() method specifies how the manager organizes fields on the screen. It retrieves the total number of fields in the manager and sets the appropriate positioning and layout for the child fields.

The layout() method invokes sublayout(). The sublayout() method controls how each child field is added to the screen by calling setPositionChild() and layoutChild() for each field that the manager contains.

```
protected void sublayout(int width, int height) {
    int x  = 0;
    int y = 0;
    Field field;
    int numberOfFields = getFieldCount();
    for (int i=0; i<numberOfFields; ++i) {
        field = getField(i);
        layoutChild(field, width, height);
        setPositionChild(field, x, y);
        field.setPosition(x,y);
        x += field.getPreferredWidth();
        y += field.getPreferredHeight();
    }
    setExtent(width,height);
}
```

Note: To set the required size for the fields, invoke `setExtent()` in `sublayout()`. If you do not invoke `setExtent()`, the field is not painted and an exception is not thrown.

Handle focus

To specify how fields should be given focus when the user rolls the trackwheel, override `nextFocus()`. The `direction` parameter indicates the direction in which the focus moves (generally, down and to the right when the trackwheel is rolled down, and up and to the left when the trackwheel is rolled up).

```
protected int nextFocus(int direction, boolean alt) {
   int index = this.getFieldWithFocusIndex();
   if(alt) {
      if(direction > 0) {
         // action to perform if trackwheel is rolled up
      } else {
         // action to perform if trackwheel is rolled down
      }
   }
   if (index == this.getFieldWithFocusIndex())
      return super.nextFocus(direction, alt);
   else
      return index;
}
```

To shift the focus to a field that is not the next field in the layout order of the manager, override `nextFocus()`. For example, `nextFocus()` is useful if you want to implement page-up and page-down functionality for the manager.

Implement subpaint()

By default, the custom manager invokes `paint()` to repaint all the fields without regard to the clipping region. If this results in unnecessary repainting, implement `subpaint()` so that fields are repainted only when the visible region changes.

Code example

Example: DiagonalManager.java

```
/**
 * DiagonalManager.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.custommenu;

import net.rim.device.api.system.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;

class DiagonalManager extends Manager {
    public DiagonalManager(long style) {
        super(style);
    }
    public int getPreferredWidth() {
        int width = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            width += getField(i).getPreferredWidth();
        }
        return width;
    }

    public int getPreferredHeight() {
        int height = 0;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            height += getField(i).getPreferredHeight();
        }
        return height;
    }

    protected void sublayout(int width, int height) {
        int x  = 0;
        int y = 0;
        Field field;
        int numberOfFields = getFieldCount();
        for (int i=0; i<numberOfFields; ++i) {
            field = getField(i);
            layoutChild( field, width, height );
            setPositionChild(field, x, y);
            x += field.getPreferredWidth();
            y += field.getPreferredHeight();
        }
        setExtent(width,height);
    }

    protected int nextFocus(int direction, boolean alt) {
        int index = this.getFieldWithFocusIndex();
        if(alt) {
            if(direction > 0) {
                // Action to perform if trackwheel is rolled up.
            }
            else {
                // Action to perform if trackwheel is rolled down.
            }
        }
        if (index == this.getFieldWithFocusIndex())
            return super.nextFocus(direction, alt);
        else
            return index;

    }
}
```

Creating lists

A `ListField` contains rows of selectable items. To enable users to select multiple items in the list, declare lists as `MULTI_SELECT`.

Implement the ListFieldCallback interface

A `ListFieldCallback` object controls all repainting tasks for the list. Each time the field is required to display an entry in the list, the necessary methods are invoked on the callback object.

To create a callback object, implement the `ListFieldCallback` interface. The system calls methods in this interface to paint a particular list row, get a particular list element, or determine the list width.

```
private class ListCallback implements ListFieldCallback {
   // the listElements vector contain the entries in the list
   private Vector listElements = new Vector();
   ...
}
```

In your implementation of `ListFieldCallback` object, implement the following three methods:

- `drawListRow()` to repaint the row when required
- `get()` to retrieve an entry from the list
- `getPreferredWidth()` to retrieve the ideal width of the list

Implement drawListRow()

To enable the field to repaint a row, implement `drawListRow()`. The graphics context that is passed into `drawListRow()` represents the entire list; accordingly, your call must specify which row to paint.

```
public void drawListRow(ListField list, Graphics g, int index, int y, int w) {
   String text = (String)listElements.elementAt(index);
   g.drawText(text, 0, y, 0, w);
}
```

Implement get()

To enable the field to retrieve an entry from the list, implement `get()`. This method returns the object contained in the row specified by `index`.

```
public Object get(ListField list, int index) {
   return listElements.elementAt(index);
}
```

Implement getPreferredWidth()

To specify the ideal width for the `List`, implement `getPreferredWidth()`. In the following implementation, `getPreferredWidth()` returns the total drawing width of the screen.

Your implementation of `getPreferredWidth()` returns a different value depending on the type of field manager. For example, if the field manager organizes fields horizontally, `getPreferredWidth()` should return the sum of the widths of each field. If the manager organizes fields vertically, `getPreferredWidth()` should return the width of the widest field.

```
public int getPreferredWidth(ListField list) {
   return Graphics.getScreenWidth();
}
```

Assigning the callback and adding the list

Create the list objects and assign the callback to the list.

Create the list objects

Create the `ListField` and `ListCallback` objects for this list. (`ListCallback` is the custom `ListFieldCallback` class created in “Implement the ListFieldCallback interface” on page 56.)

```
ListField myList = new ListField();
ListCallback myCallback = new ListCallback();
```

Set the callback

Invoke `setCallback()` to associate the `ListFieldCallback` with the `ListField`. This association enables the callback to add items to the list.

```
myList.setCallback(myCallback);
```

Add the list entries

To add entries to the list, create entries, specify an index at which to insert each entry on the `ListField` object, and then insert each `ListField` object into the `ListFieldCallback`.

```
String fieldOne = new String("Field one label");
String fieldTwo = new String("Field two label");
String fieldThree = new String("Field three label");
myList.insert(0);
myList.insert(1);
myList.insert(2);
myCallback.insert(fieldOne, 0);
myCallback.insert(fieldTwo, 1);
myCallback.insert(fieldThree, 2);
mainScreen.add(myList);
```

Code example

Example: SampleListFieldCallback.java

```
/**
 * SampleListFieldCallback.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.listfields;

import java.util.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public class SampleListFieldCallback extends UiApplication {
    private ListField myList;
    public static void main(String[] args) {
        SampleListFieldCallback app = new SampleListFieldCallback();
        app.enterEventDispatcher();
    }
    private static class ListCallback implements ListFieldCallback {
        private Vector listElements = new Vector();
        public void drawListRow(
            ListField list, Graphics g, int index, int y, int w) {
            String text = (String)listElements.elementAt(index);
            g.drawText(text, 0, y, 0, w);
        }
        public Object get(ListField list, int index) {
            return listElements.elementAt(index);
        }
        public int indexOfList(ListField list, String p, int s) {
            return listElements.indexOf(p, s);
        }
        public int getPreferredWidth(ListField list) {
            return Graphics.getScreenWidth();
        }
        public void insert(String toInsert, int index) {
            listElements.addElement(toInsert);
        }
        public void erase() {
            listElements.removeAllElements();
        }
    }
    public SampleListFieldCallback() {
        MainScreen mainScreen = new MainScreen();
        myList = new ListField();
        ListCallback myCallback = new ListCallback();
        myList.setCallback(myCallback);
        String fieldOne = "ListField one";
        String fieldTwo = "ListField two";
        String fieldThree = "ListField three";

        myList.insert(0);
        myCallback.insert(fieldOne, 0);
        myList.insert(1);
        myCallback.insert(fieldTwo, 1);
        myList.insert(2);
        myCallback.insert(fieldThree, 2);

        mainScreen.add(myList);

        pushScreen(mainScreen);
    }
}
```

Working with images

Using raw image data

Invoke `Bitmap.getARGB()` to retrieve raw image data from a specified region of a bitmap and store the data in an integer array. Applications can then manipulate the raw image data directly.

Note: The `getARGB()` method is only available on handhelds with color screens.

```
void getARGB(int[] argbData, int offset, int scanLength, int x, int y, int width,
   int height);
```

Parameter	Description
`argbData`	integer array that stores the ARGB data; each pixel is stored in 0xAARRGGBB format
`offset`	offset into the data from which to start retrieving
`scanLength`	width of a scan line within the data array
`x`	left edge of the rectangle from which to retrieve image data
`y`	top edge of the rectangle from which to retrieve image data
`width`	width of the rectangle from which to retrieve image data
`height`	height of the rectangle from which to retrieve image data

Image data is represented as one integer for each pixel, with 8 bits each for alpha (opacity), red, green, and blue values. The color components are packed into the `int` as `0xAARRGGBB`.

Retrieve image data

Initialize an integer array, and then invoke `Bitmap.getARGB()` to store the raw image data of the new or predefined bitmap in the integer array.

```
Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
int[] argb = new int[original.getWidth() * original.getHeight()];
original.getARGB(argb, 0, original.getWidth(), 0, 0, original.getWidth(),
   original.getHeight());
```

Compare two images

Invoke `Bitmap.equals()` to determine if the two bitmaps are identical.

```
if(restored.equals(original)) {
   System.out.println("Success! Bitmap renders correctly with RGB data.");
} else if(!restored.equals(original)) {
   System.out.println("Bitmap rendered incorrectly with RGB data.");
}
```

Using encoded images

The `net.rim.device.api.system.EncodedImage` class encapsulates encoded images of various formats. The handheld supports the following image formats: .gif, .png, .wbmp, and .jpeg. Only handhelds with color screens support .jpeg images.

Note: The `JPEGEncodedImage` class requires a signature that is not available to third-party developers.

Use `EncodedImage` subclasses, `PNGEncodedImage` and `WBMPEncodedImage`, to access specific properties of .png and .wbmp images, respectively. For example, `PNGEncodedImage` provides methods to retrieve the bit depth of the image, the bit depth of the alpha channel, and the color type.

An application can directly access images that are added to its project or to a library project on which it depends in the IDE.

Access an image

Save an image to the project folder or subfolder, and then add the image to the project in the IDE. Invoke `Class.getResourceAsStream()` to retrieve the image as an input stream of bytes.

```
private InputStream input;
...
try {
   input = Class.forName("com.rim.samples.docs.imagedemo.ImageDemo").
   getResourceAsStream("/images/example.png");
} catch (ClassNotFoundException e) {
   System.out.println("Class not found");
}
```

Decode an image

To encode an image, invoke `EncodedImage.createEncodedImage()`. This method creates an instance of `EncodedImage` using the raw image data in the byte array. It throws an `IllegalArgumentException` if the byte array that is provided as a parameter does not contain a recognized image format.

```
private byte[] data = new byte[2430]; // store the contents of the image file
try {
   input.read(data); // read the image data into the byte array
} catch (IOException e) {
   // handle exception
}
try {
   EncodedImage image = EncodedImage.createEncodedImage(data, 0, data.length);
} catch (IllegalArgumentException iae) {
   System.out.println("Image format not recognized.");
}
```

Tip: By default, the handheld software detects the MIME type of an image based on the image format. If the correct MIME type is not detected automatically, use the following form of `EncodedImage.createEncodedImage()` to specify a particular MIME type:

`createEncodedImage(byte[] data, createEncodedImage(byte[] data, int offset, int length, String mimeType)`

This method throws an `IllegalArgumentException` if the image format does not match the specified MIME type. Supported MIME types include: `image/gif`, `image/png`, `image/vnd.wap.wbmp`, and `image/jpeg`.

Display an encoded image

Invoke `BitmapField.setImage()` to assign the encoded image to a `BitmapField`, and then invoke `add()` to add the `BitmapField` to the screen.

```
BitmapField field = new BitmapField();
field.setImage(image);
add(field);
```

Set the decoding mode

Invoke `EncodedImage.setDecodeMode()` to set the decoding mode of the image. Provide one of the following modes as a parameter to the method:

Decoding mode	Description
DECODE_ALPHA	decodes an alpha channel, if one exists (this is the default mode)
DECODE_NATIVE	forces the bitmap to be decoded to the native bitmap type of the handheld software
DECODE_READONLY	marks the decoded bitmap as read-only

Set the scaling factor

To set the integer factor that is used to downscale an image when decoding, invoke `EncodedImage.setScale()`. The image is scaled by the inverse of the integer specified by the `scale` parameter. For example, if you set the scaling factor to 2, the image is decoded at 50% of its original size.

Code example

The ImageDemo.java sample retrieves raw data from an image that is included in its project, and then uses that raw data to recreate an `EncodedImage`.

Example: ImageDemo.java

```
/**
 * ImageDemo.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.imagedemo;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import java.io.*;

public class ImageDemo extends UiApplication {
    public static void main(String[] args) {
        ImageDemo app = new ImageDemo();
        app.enterEventDispatcher();
    }
    public ImageDemo() {
        pushScreen(new ImageDemoScreen());
    }
}

final class ImageDemoScreen extends MainScreen {
    private static final int IMAGE_SIZE = 2430;
    private InputStream input;
    private byte[] data = new byte[IMAGE_SIZE];

    public ImageDemoScreen() {
        super();
        setTitle(new LabelField("Image Demo Sample"));

        try {
            input = 
    Class.forName("com.rim.samples.docs.imagedemo.ImageDemo").getResourceAsStream(
    "/images/hellokitty.png");
        } catch (ClassNotFoundException e) {
            System.out.println("Class not found");
        }

        if(input == null) {
            System.out.println("Error: input stream is not initialized");
        } else if (input != null) {
            System.out.println("OK: input stream is initialized");
            try {
                int code = input.read(data);
                System.out.println("Total number of bytes read into buffer: " + 
    code);
            } catch (IOException e) {
                // Handle exception.
            }

            try {
                EncodedImage image = EncodedImage.createEncodedImage(data, 0, 
    data.length);
                add(new BitmapField(image.getBitmap()));
            } catch (IllegalArgumentException iae) {
                System.out.println("Image format not recognized.");
            }
        }
     }
}
```

Drawing using graphics objects

Graphics objects enable applications to perform drawing functions and rendering operations. Use the Graphics class to draw over the entire screen or on a BitmapField. If your application does not contain any fields, invoke Screen.getGraphics() to obtain the graphics context for the entire screen.

To draw over a specific BitmapField, an application obtains a graphics context for a particular field by passing the field into the Graphics constructor. When drawing over a BitmapField, the field manager passes the graphics context to the fields when the fields repaint. To perform custom drawing over a field, override the Graphics.paint() method when you extend the Field class.

The Graphics class enables you to draw shapes, such as arcs, lines, rectangles, and circles.

Use the graphics context

To draw with the Graphics class, obtain a graphics context for an individual field or the entire screen.

To obtain a graphics context for an individual field, invoke the Graphics() constructor.

```
Bitmap surface = new Bitmap(100, 100);
BitmapField surfaceField = new BitmapField(surface);
Graphics graphics = new Graphics(surface);
```

To obtain a graphics context for the entire screen, invoke Screen.getGraphics().

```
Graphics graphics = Screen.getGraphics();
```

To draw using any graphics context, your methods must perform their drawing functions within the boundaries of the field or screen.

```
graphics.fillRect(10, 10, 30, 30);
graphics.drawRect(15, 15, 30, 30);
```

If your graphics context does not apply to the entire screen, add the `BitmapField` to the screen.

```
mainScreen.add(surfaceField);
```

Implementing the DrawStyle interface

The `DrawStyle` interface provides styles that are used by `Graphics` and `Field` objects. Implementing `DrawStyle` enables you to create an interface that is consistent with standard BlackBerry user interfaces. If you are extending the `Field` class to create a custom field, your code should accept the appropriate styles so that it acts similarly to standard BlackBerry applications.

`DrawStyle` is applied to fields as style parameters, as in the following example:

```
ButtonField buttonField = new ButtonField(DrawStyle.ELLIPSIS);
```

You can use `DrawStyle` elements in the following objects:

- `BitmapField`
- `ButtonField`
- `DateField`
- `Graphics`
- `LabelField`
- `ObjectListField`

Drawing in color

Drawing in color is only applicable to handhelds with color screens. To determine whether handhelds provide color display, invoke `Graphics.isColor()`. To determine the number of colors that handhelds support, invoke `Graphics.numColors()`.

Set alpha values

The global alpha value determines the transparency of pixels in the drawing area, where 0 (0x0000) is completely transparent (invisible) and 255 (0x00FF) is fully opaque. To set or get the global alpha value, invoke `Graphics.setGlobalApha()` or `Graphics.getGlobalAlpha()`.

Note: The alpha value is used only for certain raster operations. Text and drawing operations do not use the alpha value.

Determine raster operation support

To determine if a `Graphics` object supports a particular raster operation, invoke `Graphics.isRopSupported( int rop )`.

Constant	Raster operation
ROP_CONST_GLOBALALPHA	blends the constant foreground color using a constant global alpha value with destination pixels

Constant	Raster operation
ROP_SRC_GLOBALALPHA	blends a source bitmap using a constant global alpha value with destination pixels

Draw a path

To draw a set of shaded, filled paths, invoke `Graphics.drawShadedFilledPath()`:

```
public void drawShadedFilledPath(int[] xPts, int[] yPts, byte[] pointTypes, int[]
   colors, int[] offsets);
```

Constant	Description
`xPts`	This ordered list defines x values for each vertex in the paths.
`yPts`	This ordered list defines y values for each vertex in the paths.
`pointTypes`	Specify one of the following constants for each (x,y) point defined. If `pointTypes` is `null`, all points default to `Graphics.CURVEDPATH_END_POINT`. • `Graphics.CURVEDPATH_END_POINT` • `Graphics.CURVEDPATH_QUADRATIC_BEZIER_CONTROL_POINT` • `Graphics.CURVEDPATH_CUBIC_BEZIER_CONTROL_POINT`
`colors`	This ordered list defines color values for each vertex, in 0x00RRGGBB format. If `null`, a solid filled path is drawn in the current foreground color.
`offsets`	This list defines the start of each path in the `xPts` and `yPts` data arrays. `null` indicates a single path; a path that begins at point (`xPts[offsets[i]],yPts[offsets[i]]`) and ends at point (`xPts[offsets[i+1]]-1,yPts[offsets[i+1]]-1`).

The following example draws a path that blends from blue to red.

```
Bitmap surface = new Bitmap(240, 160);
BitmapField surfaceField = new BitmapField(surface);
add(surfaceField);
Graphics graphics = new Graphics(surface);
int[] X_PTS = { 0, 0, 240, 240 };
int[] Y_PTS = { 20, 50, 50, 20 };
int[] drawColors = { 0x0000CC, 0x0000CC, 0xCC0000, 0xCC0000 };
try {
   graphics.drawShadedFilledPath(X_PTS, Y_PTS, null, drawColors, null);
} catch (IllegalArgumentException iae) {
   System.out.println("Bad arguments.");
}
```

Use drawing styles

To turn a drawing style on or off, invoke `Graphics.setDrawingStyle(int drawStyle, boolean on)`, where `on` specifies whether to turn the style on (true) or off (false). To determine if a drawing style is set, invoke `Graphics.isDrawingStyleSet(int drawStyle)`.

Constant	Description
DRAWSTYLE_AALINES	style for anti-aliased rendering of lines; used by `setDrawingStyle()` and `isDrawingStyleSet()`
DRAWSTYLE_AAPOLYGONS	style for anti-aliased rendering of polygons; used by `setDrawingStyle()` and `isDrawingStyleSet()`
DRAWSTYLE_FOCUS	style set by the framework when painting is being done for focus drawing
DRAWSTYLE_SELECT	style set by the framework when painting is being done for selection drawing

Use monochrome bitmap fields like a stamp

The `STAMP_MONOCHROME` option enables applications to use monochrome bitmaps like a stamp by rendering the non-transparent region in color. This option applies to bitmaps that are 1 bit and have alpha defined.

```
BitmapField field = new BitmapField(original, BitmapField.STAMP_MONOCHROME);
```

Draw an image from raw data

1. Create an empty bitmap. In this example, the type and size are copied from an existing bitmap.
2. Create a `Graphics` object using the newly created bitmap as the drawing surface.
3. Invoke `Graphics.rawRGB()` to draw a new image using raw data retrieved from the original.

```
Bitmap restored = new Bitmap(original.getType(), original.getWidth(),
   original.getHeight());
Graphics graphics = new Graphics(restored);
try {
   graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0, restored.getWidth(),
      restored.getHeight());
} catch(Exception e) {
   System.out.println("Error occurred during drawing: " + e);
}
```

Using bitmap types

Note: The following details on bitmap types are provided for information only. Applications should not rely on the actual bit format of bitmaps as formats are subject to change in future releases of BlackBerry Handheld Software.

To determine the `Bitmap` type, invoke `Bitmap.getType()`. This method returns one of the following constants:

Bitmap Type	Description
COLUMNWISE_MONOCHROME	Data is stored in columns, with 1 bit for each pixel: 0 is white and 1 is black. Uppermost pixels are in the less significant bits in a byte, and lower numbered bytes contain the uppermost pixels within a column.
ROWWISE_MONOCHROME	Data is stored in rows, with 1 bit for each pixel: 0 is black and 1 is white. Each row is a multiple of 4 bytes in width. Leftmost pixels are in the less significant bits in a byte, and lower numbered bytes contain the leftmost pixels within a row.
ROWWISE_16BIT_COLOR	Data is stored in rows, with 2 bytes for each pixel: 0 is black and a 0xffff (65535) is white. Each row is a multiple of 4 bytes in width.

- On handhelds with monochrome screens, data is ordered in columns, so `Bitmap.getType()` returns `COLUMNWISE_MONOCHROME`. The first 2 bytes represent the first 16 pixels in the first column of the bitmap.
- On handhelds with color screens, data is ordered in rows, so `Bitmap.getType()` returns `ROWWISE_MONOCHROME` for monochrome images or `ROWWISE_16BIT_COLOR` for color images. In a monochrome image, the first 2 bytes represent the first 16 pixels in the first row of the bitmap, going from left to right. In a color image, the first 2 bytes represent the first pixel.

The following two forms of the `Bitmap` constructor enable you to specify a type parameter:

- `Bitmap(int type, int width, int height)`
- `Bitmap(int type, int width, int height, byte[] data)`

To retrieve the default bitmap type for the handheld, invoke `Bitmap.getDefaultType()`.

Code example

The DrawDemo.java sample retrieves raw data from a predefined bitmap image, and then draws a new bitmap using the data. It then displays the original and restored images.

Example: DrawDemo.java

```
/*
 * DrawDemo.java
 * Copyright (C) 2002-2004 Research In Motion Limited.
 */

package com.rim.samples.docs.drawing;

import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

public class DrawDemo extends UiApplication {

    public static void main(String[] args) {
        DrawDemo app = new DrawDemo();
        app.enterEventDispatcher();
    }

    public DrawDemo() {
        pushScreen(new DrawDemoScreen());
    }
}

final class DrawDemoScreen extends MainScreen {
    public DrawDemoScreen() {
        super();

        LabelField title = new LabelField("UI Demo", LabelField.USE_ALL_WIDTH);
        setTitle(title);

        Bitmap original = Bitmap.getPredefinedBitmap(Bitmap.INFORMATION);
        Bitmap restored = new Bitmap(original.getType(), original.getWidth(),
                        original.getHeight());

        Graphics graphics = new Graphics(restored);

        // Retrieve raw data from original image.
        int[] argb = new int[original.getWidth() * original.getHeight()];
        original.getARGB(argb, 0, original.getWidth(), 0, 0, original.getWidth(),
                        original.getHeight());

        // Draw new image using raw data retrieved from original image.
        try {
            graphics.drawRGB(argb, 0, restored.getWidth(), 0, 0,
   restored.getWidth(),
                restored.getHeight());
        } catch(Exception e) {
            System.out.println("Error occurred during drawing: " + e);
        }


        if(restored.equals(original)) {
            System.out.println("Success! Bitmap renders correctly with RGB data.");
        } else if(!restored.equals(original)) {
            System.out.println("Bitmap rendered incorrectly with RGB data.");
        }

        BitmapField field1 = new BitmapField(original,
   BitmapField.STAMP_MONOCHROME);
        BitmapField field2 = new BitmapField(restored);
        add(new LabelField("Original bitmap: "));
        add(field1);
        add(new LabelField("Restored bitmap: "));
        add(field2);
    }
}
```

Listening for changes to UI objects

UI EventListeners enable applications to respond to a change to a UI object. There are three types of UI event listeners:

Listener	Description
FieldChangeListener	notifies when a field property changes
FocusChangeListener	notifies when a field gains or loses focus
ScrollChangeListener	notifies when the horizontal or vertical scrolling value of a manager changes

Listen for field property changes

To monitor changes to fields, implement the FieldChangeListener interface and assign it to a field by invoking setChangeListener().

```
private class FieldListener implements FieldChangeListener {
   public void fieldChanged(Field field, int context) {
      if (context != FieldChangeListener.PROGRAMMATIC) {
         // perform action if user changed field
      } else {
         // perform action if application changed field
      }
   }
}
// ...
FieldListener myFieldChangeListener = new FieldListener()
myField.setChangeListener(myFieldChangeListener);
```

Listen for focus changes

To monitor changes in focus between fields, assign them a FocusChangeListener by implementing FocusChangeListener and invoking setChangeListener(). A FocusChangeListener considers the gain, loss, or change of focus in relation to a particular field.

Your implementation of `FocusChangeListener` should specify what action to take when the field gains, loses, or changes the focus by implementing `focusChanged()`.

```
private class FocusListener implements FocusChangeListener {
   public void focusChanged(Field field, int eventType) {
      if (eventType == 0) {
         // perform action when this field gains the focus
      }
      if (eventType == 1) {
         // perform action when the focus changes for this field
      }
      if (eventType == 2) {
         // perform action when this field loses the focus
      }
   }
}
FocusListener myFocusChangeListener = new FocusListener();
myField.setChangeListener(myFocusChangeListener);
```

Listen for scroll events

To enable your field manager to manage scroll events, implement the `ScrollChangeListener` interface and then invoke `setScrollListener()`. When the horizontal or vertical (or both) scrolling values change, the `scrollChanged()` method passes in the new values.

Tip: Typically, listening for scrolling changes is unnecessary because your application can listen for focus changes to fields; however, `ScrollChangeListener` can be useful in a game implementation.

To assign a listener to a field, invoke `setScrollListener()` on the field manager.

```
private class ScrollListener implements ScrollChangeListener {
   scrollChanged(Manager manager, int newHoriztonalScroll, int newVerticalScroll){
      // perform action
   }
}
ScrollListener myScrollChangeListener = new ScrollListener();
myManager.setScrollListener(myScrollChangeListener);
```

Supporting media content

PME content

BlackBerry handhelds support rich media content in PME format.

Content developers can create PME content using Plazmic Content Developer's Kit for BlackBerry® ™. This tool, and accompanying documentation, is available from the Plazmic web site at www.plazmic.com.

The Media Engine APIs (in the `net.rim.plazmic.mediaengine` and `net.rim.plazmic.mediaengine.io` packages) enable applications to retrieve and play PME content that is stored on the handheld or on the network.

Note: The Media Engine APIs support the media type `application/x-vnd.rim.pme`. Web servers must set the MIME type for .pme files to `application/x-vnd.rim.pme`.

Overview of PME APIs

The following three main classes (in the `net.rim.plazmic.mediaengine` package) provide the ability to load and play PME media content:

Class	Description
`MediaManager`	provides methods for loading content from local storage or the network
`MediaPlayer`	provides methods for playing PME media
`MediaException`	provides exception codes for errors specific in retrieving or playing media

Media loading

The Media Engine API enables applications to load media content using one of the following four protocols:.

Protocol	Description
`http://`	The http protocol downloads content from a web server on the network using an HTTP connection. This protocol requires a BlackBerry Enterprise Server™ with the Mobile Data Service enabled.
`https://`	The https protocol downloads content from a web server on the network using a secure HTTP (HTTPS) connection. This protocol requires a BlackBerry Enterprise Server with the Mobile Data Service enabled.
`jar:///`*`<pme_file>`*	The jar protocol loads content from a .jar file that is stored locally on the handheld. `jar:///sample.pme` Note that the leading slash is required. In the IDE, the .jar file must be added to the same project as the calling application or to a library project on which the application depends.

Protocol	Description
cod:// *<module><pme_file>*	The cod protocol loads content from a .cod file (module) that is stored locally on the handheld. cod://mediasample/sample.pme

To use another protocol, implement a custom Connector. See " Creating custom connections" on page 79 for more information.

Playback states

To retrieve the current state of the MediaPlayer, invoke MediaPlayer.getState().

State	Description
UNREALIZED	The MediaPlayer is not ready to play media. To move to the REALIZED state, invoke MediaPlayer.setMedia().
REALIZED	The MediaPlayer is ready to play media. To start playback and move to the STARTED state, invoke MediaPlayer.start().
STARTED	The MediaPlayer is playing media. To stop playback and return to the REALIZED state, invoke MediaPlayer.stop().

Exceptions

Methods on MediaEngine and MediaManager classes throw a MediaException that includes a standard HTTP response code or one of the following exception codes. To retrieve the error code associated with an exception, invoke MediaException.getCode().

Exception code	Description
INVALID_HEADER	The media format is invalid.
REQUEST_TIMED_OUT	The request has timed out.
INTERRUPTED_DOWNLOAD	The download was cancelled by the application invoking MediaManager.cancel().
UNSUPPORTED_TYPE	The media format (MIME type) is unsupported.
UPGRADE_PLAYER	The current version of the media engine is not compatible with the requested content.
UPGRADE_MEDIA	The current version of the media engine no longer supports the requested content.
CHECKSUM_MISMATCH	The checksum verification failed, so the media cannot be read.
OUT_OF_BOUNDS	An array index is out of bounds, or the application tried to read from an input stream after the end of the file.

Events

The MediaListener interface enables applications to receive and respond to the following events:

Event	Description
MEDIA_REQUESTED	Media has been requested for loading; occurs when an animation automatically requests new content or when the user clicks a hyperlink for media content.
MEDIA_REALIZED	Media has been created for playback; occurs when MediaManager.createMediaLater() is invoked.
MEDIA_COMPLETE	Media has been loaded and played successfully.
MEDIA_IO	Media is being loaded.

See " Listening for media engine events" on page 73 for more information.

Playing media content

To retrieve PME content on handhelds or networks, use methods on the `MediaManager`. To play PME content that has been downloaded to handhelds, use methods on the `MediaPlayer` class.

Download content

To download PME content, create a `MediaManager` object, and then invoke `MediaManager.createMedia()`.

```
MediaManager manager = new MediaManager();
try {
   Object media = manager.createMedia("http://webserver/sample.pme");
} catch (IOException ioe) {
   System.out.println("Error: requested content was not downloaded.");
} catch (MediaException me) {
   System.out.println("Error: " + me.getCode());
}
```

Notes: The following default protocols are supported: http://, https://, jar:// and cod://. See " Media loading" on page 69 for more information.

The first time that you invoke `MediaManager.createMedia()`, the URL must be absolute, unless you first invoke `MediaManager.setProperty("URI_BASE", <base_url>)` to set a base URL. When you invoke `createMedia()` subsequently, the previous URL is used as the base.

Play PME content

Set the PME object for playback

To set the PME object for playback, invoke `MediaPlayer.setMedia()`.

```
MediaPlayer player = new MediaPlayer();
try {
   player.setMedia(media);
} catch (MediaException me) {
   System.out.println("Error: requested content type is not supported.");
}
```

Display PME content

To return a UI object that can display the PME content, invoke `MediaPlayer.getUI()`. Cast the object that `getUI()` returns as a `Field` and add it to a `Screen` for display.

```
add((Field)player.getUI());
```

Start playing PME content

To start playing the downloaded content, invoke `MediaPlayer.start()`.

```
if(player.getState() == MediaPlayer.REALIZED) {
   try {
      player.start();
   } catch(MediaException me) {
      System.out.println("Error occurred during media playback: " +
        me.getCode() + me.getMessage());
    }
}
```

Tip: Verify the current state of the `MediaPlayer` before you invoke `MediaPlayer.start()`. The `start()` method throws an exception if the media player is not in the REALIZED state.

Code example

The MediaSample.java sample retrieves a PME file from a web server and displays it.

Example: MediaSample.java

```
/**
 * MediaSample.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

import net.rim.plazmic.mediaengine.*;

public class MediaSample extends UiApplication {

    public static void main(String[] args) {
        MediaSample app = new MediaSample();
        app.enterEventDispatcher();
    }
    public MediaSample() {
        pushScreen(new MediaSampleScreen());
    }
    final static class MediaSampleScreen extends MainScreen {
        public MediaSampleScreen() {
            super();
            LabelField title = new LabelField("Media Sample", LabelField.ELLIPSIS
            | LabelField.USE_ALL_WIDTH);
            setTitle(title);

            MediaPlayer player = new MediaPlayer();
            MediaManager manager = new MediaManager();

            try {
               Object media = manager.createMedia("http://webserver/SVGFILE.pme");
                player.setMedia(media);
            } catch (IOException ioe) {
            } catch (MediaException me) {
                System.out.println("Error during media loading: ");
                System.out.println(me.getCode());
                System.out.println(me.getMessage());
            }
            add((Field)player.getUI());
            try {
                player.start();
            } catch(MediaException me) {
                System.out.println("Error occured during media playback: ");
                System.out.println(me.getCode());
                System.out.println(me.getMessage());
            }
        }
    }
}
```

Listening for media engine events

The `MediaListener` interface enables applications to register to receive media engine events. Applications can register listeners on both `MediaPlayer` and `MediaEngine` objects.

When an application implements the listener, it can perform the following actions:

- provide information on the status of content downloads
- download content in the background and play it when it arrives
- download content that is requested automatically by an animation

The `MediaListener` interface includes one method, the listen method.

```
public void mediaEvent(Object sender, int event, int eventParam, Object data);
```

Parameter	Description
`sender`	This parameter refers to the object that sent the event, such as a `MediaPlayer` or `MediaManager` object.
`event`	This parameter is one of the following events: • `MEDIA_REQUESTED`: Sent when new content is requested. • `MEDIA_COMPLETE`: Sent when all scheduled media actions have been completed. • `MEDIA_REALIZED`: Sent by a `MediaManager` to return downloaded media. • `MEDIA_IO`: Sent by the `MediaLoader` to provide information on download progress or status.
`eventParam`	Do not use this parameter as it may receive an arbitrary value. It exists in order to provide a consistent interface for additional events.
`data`	When `event` is `MEDIA_REQUESTED`, `data` refers to the requested URL as a `String` object. When `event` is `MEDIA_REALIZED`, `data` refers to the media object that was created. When `event` is `MEDIA_IO`, `data` refers to a `net.rim.plazmic.mediaengine.io.LoadingStatus` object.

Implement the listener

Implement the `MediaListener` interface, including `mediaEvent()`. The following example uses a `switch` statement to handle possible media events.

```
public final class MediaListenerImpl implements MediaListener {
    public void mediaEvent(Object sender, int event, int eventParam, Object data) {
        switch(event) {
            case MEDIA_REQUESTED: break;
            case MEDIA_COMPLETE: break;
            case MEDIA_REALIZED: break;
            case MEDIA_IO: break;
        }
    }
}
```

Register the listener

To register your listener, invoke `addMediaListener()` on the `MediaPlayer` and `MediaManager` objects.

```
private MediaListenerImpl _listener = new MediaListenerImpl();
private MediaPlayer player = new MediaPlayer();
private MediaManager manager = new MediaManager();
player.addMediaListener(_listener);
manager.addMediaListener(_listener);
```

Load content in the background

When you implement `MediaListener`, you can download PME content in the background (asynchronously), and then play the content when the download is complete.

Invoke `MediaManager.createMediaLater()` to download content for future playback.

Note: Unlike `createMedia()`, `createMediaLater()` does not return an `Object` with the media content.

In `MediaListener.mediaEvent()`, add code to handle the `MEDIA_REALIZED` event that occurs when the requested content has been loaded successfully. To register the content that is specified in the data parameter, invoke `MediaPlayer.setMedia(data)`. To start playback, invoke `MediaPlayer.start()`.

```
manager.createMediaLater("http://webserver/sample.pme");
public void mediaEvent(Object sender, int event, int eventParam, Object data) {
    switch(event) {
        ...
        case MEDIA_REALIZED:
            try {
                player.setMedia(data);
                player.start();
            } catch(MediaException me) {
                System.out.println("Error playing media" + me.getCode() +
                me.getMessage());
            }
        break;
    }
}
```

Track download progress

To obtain information about download progress, use the `net.rim.plazmic.mediaengine.io.LoadingStatus` class. This class includes methods that enable you to retrieve the media content type, the total number of bytes, the number of bytes read, and the source URL of the content.

Status	Description
LOADING_STARTED	Loading has started.
LOADING_READING	The data stream is being parsed.
LOADING_FINISHED	The media has been loaded successfully.
LOADING_FAILED	The media has not been loaded successfully. • To obtain detailed error codes, invoke `getCode()`. See " Exceptions" on page 70 for more information. • To obtain the exception message, invoke `getMessage()`.

In the implementation of `mediaEvent()`, when the `MEDIA_IO` event occurs, cast the `Object` in the `data` parameter to a `LoadingStatus` object.

Use a `switch` statement to handle each status. Invoke `LoadingStatus.getStatus()` to retrieve the download status.

For each normal status, print a message to the console.

For the `LOADING_FAILED` status, perform the following actions:

- Invoke `LoadingStatus.getCode()` to retrieve the error code.
- Invoke `LoadingStatus.getMessage()` to retrieve the detailed message.
- Invoke `LoadingStatus.getSource()` to retrieve the URL string of the content.

```
public void mediaEvent(Object sender, int event, int eventParam, Object data) {
    switch(event) {
        ...
        case MEDIA_IO: {
            LoadingStatus s = (LoadingStatus)data;
        }
        ...
        break;
    }
    break;
    ...
    switch(s.getStatus()) {
        case LoadingStatus.LOADING_STARTED:
            System.out.println("Loading in progress");
        break;
        case LoadingStatus.LOADING_READING:
            System.out.println("Parsing in progress");
        break;
        case LoadingStatus.LOADING_FINISHED:
            System.out.println("Loading completed");
        break;
        case LoadingStatus.LOADING_FAILED:
            String errorName = null;
            int code = s.getCode();
            switch (code) {
                case MediaException.INVALID_HEADER:
                    errorName =  "Invalid header" + "\n" +  s.getSource();
                break;
                case MediaException.REQUEST_TIMED_OUT:
                    errorName = "Request timed out" + "\n" +
                    s.getSource();
                break;
                case MediaException.INTERRUPTED_DOWNLOAD:
                break;
                case MediaException.UNSUPPORTED_TYPE:
                    errorName = "Unsupported type" + s.getMessage() + "\n" +
                    s.getSource();
                break;
                default: {
                    if (code > 200) {
                        // A code > 200 indicates an HTTP error
                        errorName = "URL not found";
                    } else {
                        // default unidentified error
                        errorName = "Loading Failed";
                }
                errorName += "\n" + s.getSource() + "\n" + s.getCode()
                + ": " + s.getMessage();
                break;
            }
        }
        System.out.println(errorName);
        break;
    } //end switch s.getStatus
    break;
}
```

Code example

The MediaSample2.java sample implements a listener to download media content in the background and display the download status to the console.

Example: MediaSample2.java

```
/**
 * MediaSample2.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;

import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class MediaSample2 extends UiApplication {
 private MediaPlayer player = new MediaPlayer();
 private MediaManager manager = new MediaManager();
 private MediaListenerImpl _listener = new MediaListenerImpl();

 private MediaSample2Screen _screen;

 public static void main(String[] args) {
      MediaSample2 app = new MediaSample2();
      app.enterEventDispatcher();
 }

 public MediaSample2() {
     _screen = new MediaSample2Screen();
     pushScreen(_screen);
 }

 public final class MediaListenerImpl implements MediaListener {
     public void mediaEvent(Object sender, int event, int eventParam, Object 
data) {
         switch(event) {
             case MEDIA_REQUESTED:
                 System.out.println("Media requested");
                 break;
             case MEDIA_COMPLETE:
                 System.out.println("Media completed");
                 break;
             case MEDIA_REALIZED:
                 try {
                     player.setMedia(data);
                     player.start();
                 }
                 catch(MediaException me) {
                     System.out.println("Error during media loading: " + 
me.getCode() + me.getMessage());
                 }
                 break;
             case MEDIA_IO: {
                 LoadingStatus s = (LoadingStatus)data;
                 switch(s.getStatus()) {
                     case LoadingStatus.LOADING_STARTED:
                         System.out.println("Loading in progress");
                         break;
                     case LoadingStatus.LOADING_READING:
                         System.out.println("Parsing in progress");
                         break;
                     case LoadingStatus.LOADING_FINISHED:
                         System.out.println("Loading completed");
                         break;
                     case LoadingStatus.LOADING_FAILED:
                         String errorName = null;
                         int code = s.getCode();
                         switch (code) {
                             case MediaException.INVALID_HEADER:
                                 errorName =  "Invalid header" + "\n" + 
s.getSource();
                                 break;
```

```
                                case MediaException.REQUEST_TIMED_OUT:
                                    errorName = "Request timed out" + "\n" +
s.getSource();
                                    break;

                                case MediaException.INTERRUPTED_DOWNLOAD:
                                    break;

                                case MediaException.UNSUPPORTED_TYPE:
                                   errorName = "Unsupported type" + s.getMessage()
+ "\n" + s.getSource();
                                    break;

                                default: {
                                    if (code > 200) {
                                        // A code > 200 indicates an HTTP error.
                                        errorName = "URL not found";
                                    }
                                    else {
                                        // Default unidentified error.
                                        errorName = "Loading Failed";
                                    }
                                    errorName += "\n" + s.getSource() + "\n"
                                        + s.getCode() + ": " + s.getMessage();
                                    break;
                                }

                            }
                            System.out.println(errorName);
                            break;
                    } // End switch s.getStatus().
                    break;
                }
            }
        }
    }
    final class MediaSample2Screen extends MainScreen {
        public MediaSample2Screen() {
            super();
            LabelField title = new LabelField("Media Sample", LabelField.ELLIPSIS
                | LabelField.USE_ALL_WIDTH);
            setTitle(title);

            player.addMediaListener(_listener);
            manager.addMediaListener(_listener);

            // Change this to the location of a test .pme file.
            manager.createMediaLater("http://test.rim.com/SVGBS0001.pme");
            add((Field)player.getUI());
        }
    }
}
```

Creating custom connections

MediaManager uses a Connector object to load media and open an input stream. The default Connector supports the following protocols: http://, https://, jar://, and cod://. To add support for a custom protocol or to override default behavior, create a custom Connector by implementing the net.rim.plazmic.mediaengine.io.Connector interface.

Method signature	Implementation
InputStream getInputStream(String uri, ConnectionInfo info)	Implement this method to return an input stream to read content from the specific URI.
void releaseConnection(ConnectionInfo info)	Implement this method to release the connection. MediaManager invokes this method to inform a Connector that it can release the connection.
void setProperty(String name, String value)	Implement this method to set connector-specific properties.

Implement a custom connector

To perform processing for a custom protocol, implement the Connector interface, including getInputStream() . To handle a standard protocol, invoke the default Connector.

To set specific properties, implement setProperty(String name, String value). In this example, the connector does not have to set any specific properties, so the implementation of setProperty() invokes setProperty() on the default Connector.

```
public class SampleConnector implements Connector {
    Connector delegate; // the default Connector
    SampleConnector(Connector delegate) {
        this.delegate = delegate;
    }
    public InputStream getInputStream(String uri, ConnectionInfo info)
        throws IOException, MediaException {
        InputStream input = null;
        if (uri.startsWith("myprotocol://")) {
            //perform special tasks
            info.setConnection(new MyProtocolConnection());
            info.setContentType("application/x-vnd.rim.pme");
            // openMyInputStream is a custom method that opens
            //stream for "myprotocol://"
            input = openMyInputStream(uri);
        } else {
            input = delegate.getInputStream(uri, info);
        }
        return input;
    }
    public void releaseConnection(ConnectionInfo info)
        throws IOException, MediaException {
        Object o = info.getConnection();
        if (o instanceof MyProtocolConnection) {
            ((MyProtocolConnection)o).close(); // perform cleanup
        } else {
            delegate.releaseConnection(info);
        }
    }
    public void setProperty(String property, String value) {
        delegate.setProperty(property, value);
    }
}
```

Register a custom connector

In your main method, invoke `MediaManager.setConnector()` to register your custom connector.

```
MediaManager manager = new MediaManager();
manager.setConnector(new CustomPMEConnector(manager.getDefaultConnector()));
```

Code example

The CustomPMEConnector.java sample provides a framework for implementing a custom connector.

Example: CustomPMEConnector.java

```
/*
 * CustomPMEConnector.java
 * Copyright (C) 2003-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.mediasample;

import java.io.*;
import net.rim.plazmic.mediaengine.*;
import net.rim.plazmic.mediaengine.io.*;

public class CustomPMEConnector implements Connector {

    private Connector delegate;
    private InputStream input;

    CustomPMEConnector(Connector delegate) {
        this.delegate = delegate;
    }
    public InputStream getInputStream(String uri, ConnectionInfo info)
        throws IOException, MediaException {
        if (uri.startsWith("myprotocol://")) {
            // Perform special tasks.
            info.setConnection(new MyProtocolConnection());
            info.setContentType("application/x-vnd.rim.pme");
            // OpenMyInputStream() is a custom method that opens
            //stream for "myprotocol://"
            input = openMyInputStream(uri);
        } else {
            input = delegate.getInputStream(uri, info);
        }
        return input;
    }

    private InputStream openMyInputStream(String uri) {
        InputStream input = null;
```

```
            // @todo: open stream here
            return input;
        }
        public void releaseConnection(ConnectionInfo info)
            throws IOException, MediaException {
            Object o = info.getConnection();
            if (o instanceof MyProtocolConnection) {
                ((MyProtocolConnection)o).close(); // Perform cleanup.
            } else {
                delegate.releaseConnection(info);
            }
        }
        public void setProperty(String property, String value) {
            delegate.setProperty(property, value);
        }
        // Inner class that defines the connection class.
        public static class MyProtocolConnection {
            public MyProtocolConnection() {
                // ...
            }
            public void close() {
                // ...
            }
        }
    }
```

Connecting to networks

HTTP and socket connections

Although you can implement HTTP over a socket connection, it is preferable to use an HTTP connection because socket connections do not support the Mobile Data Service features, such as push. It is also preferable to use HTTP connections because applications that use socket connections typically require significantly more bandwidth than those that use HTTP connections.

Tip: If you use sockets, design the application to accommodate intermittent connections to the wireless network. For example, the application should re-open the connection if an error occurs.

Using HTTP connections

Note: The BlackBerry Internet Service Browser does not allow Java applications to initiate HTTP, HTTPS, or TCP connections.

Open an HTTP connection

To open an HTTP connection, `invoke Connector.open()`, specifying `http` as the protocol. Cast the returned object as an `HTTPConnection` or a `StreamConnection` object. An `HttpConnection` is a `StreamConnection` that provides access to specific HTTP functionality, including headers and other HTTP resources.

```
HttpConnection conn = null;
String URL = "http://www.myServer.com/myContent";
conn = (HttpConnection)Connector.open(URL);
```

Set the HTTP request method

To set the HTTP request method (`GET` or `POST`), invoke `HttpConnection.setRequestMethod()`.

```
conn.setRequestMethod(HttpConnection.POST);
```

Set or retrieve header fields

To set or retrieve header fields for HTTP request or HTTP response messages, invoke `getRequestProperty() or setRequestProperty()` on the `HttpConnection`.

```
conn.setRequestProperty("User-Agent", "BlackBerry/3.2.1");
String lang = conn.getRequestProperty("Content-Language");
```

Send and receive data over HTTP

To send and receive data, acquire input and output streams by invoking `openInputStream()` and `openOutputStream()` on the `HTTPConnection`.

```
InputStream in = conn.openInputStream();
OutputStream out = conn.openOutputStream();
```

Code example

The HttpFetch.java example uses an HTTP connection to retrieve data. It performs the following steps:

1. Creates a connection thread.
2. Defines a method to retrieve data.
3. Defines a method to display data to the user.
4. Defines a method to exit the application.
5. Creates the application constructor.

Note: The HTTPFetch.java example requires you to create resource files in the application project and define the required resource keys. See " Localizing applications" on page 109 for more information.

Example: HTTPFetch.java

```
/**
 * HTTPFetch.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.httpfetch;

import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.io.*;
import com.rim.samples.docs.baseapp.*;

public class HTTPFetch extends BaseApp implements HTTPFetchResource {
    // Constants.
    private static final String SAMPLE_PAGE = "http://localhost/testpage/
   sample.txt";
    private static final String[] HTTP_PROTOCOL = {"http://", "http:\\"};

    // Members.
    private MainScreen _mainScreen;
    private RichTextField _content;

    /**
```

```
* Send and receive data over the network on a
* separate thread from the main thread of your application.
*/
ConnectionThread _connectionThread = new ConnectionThread();

//statics
private static ResourceBundle _resources = ResourceBundle.getBundle(
HTTPFetchResource.BUNDLE_ID, HTTPFetchResource.BUNDLE_NAME);

public static void main(String[] args) {
    HTTPFetch theApp = new HTTPFetch();
    theApp.enterEventDispatcher();
}
/**
* The ConnectionThread class manages the HTTP connection.
* Fetch operations are not queued, but if a second fetch request
* is made while a previous request is still active,
* the second request stalls until the previous request completes.
*/
private class ConnectionThread extends Thread {
    private static final int TIMEOUT = 500; //ms

    private String _theUrl;

    /* The volatile keyword indicates that because the data is shared,
    * the value of each variable must always be read and written from memory,
    * instead of cached by the VM. This technique is equivalent to wrapping
    * the shared data in a synchronized block, but produces less overhead.
    */
    private volatile boolean _start = false;
    private volatile boolean _stop = false;

    /**
    * Retrieve the URL. The synchronized keyword ensures that only one
    * thread at a time can call this method on a ConnectionThread object.
    */
    public synchronized String getUrl() {
        return _theUrl;
    }

    /**
    * Fetch a page. This method is invoked on the connection thread by
    * fetchPage(), which is invoked in the application constructor when
    * the user selects the Fetch menu item.
    */
    public void fetch(String url) {
        _start = true;
        _theUrl = url;
    }

    /**
     *  Close the thread. Invoked when the application exits.
     */
    public void stop() {
        _stop = true;
    }

    /**
    * Open an input stream and extract data. Invoked when the thread
    * is started.
    */
    public void run() {
        for(;;) {
            // Thread control.
            while( !_start && !_stop) {
                // No connections are open for fetch requests,
                // but the thread has not been stopped.
                try {
                    sleep(TIMEOUT);
                } catch (InterruptedException e) {
                    System.err.println(e.toString());
                }
            }
            // Exit condition.
            if ( _stop ) {
                return;
            }
            /* Ensure that fetch requests are not missed
             * while received data is processed.
             */
            synchronized(this) {
                // Open the connection and extract the data.
                StreamConnection s = null;
                try {
                    s = (StreamConnection)Connector.open(getUrl());
                    InputStream input = s.openInputStream();
                    // Extract data in 256 byte chunks.
                    byte[] data = new byte[256];
                    int len = 0;
                    StringBuffer raw = new StringBuffer();
                    while ( -1 != (len = input.read(data)) ) {
                        raw.append(new String(data, 0, len));
                    }
                    String text = raw.toString();

                    updateContent(text);

                    input.close();
                    s.close();
                } catch (IOException e) {
                    System.err.println(e.toString());
                    // Display the text on the screen.
                    updateContent(e.toString());
                }
                // Reset the start state.
                _start = false;
            }
        }
    }
}
```

```
    // Constructor.
    public HTTPFetch() {
        _mainScreen = new MainScreen();
        _mainScreen.setTitle(new LabelField(
        _resources.getString(APPLICATION_TITLE), LabelField.ELLIPSIS
        | LabelField.USE_ALL_WIDTH));
        _mainScreen.add(new SeparatorField());
        _content = new RichTextField(
        _resources.getString(HTTPDEMO_CONTENT_DEFAULT));
        _mainScreen.add(_content);
        _mainScreen.addKeyListener(this);
        _mainScreen.addTrackwheelListener(this);

        // Start the helper thread.
        _connectionThread.start();
        pushScreen(_mainScreen);
        fetchPage(SAMPLE_PAGE);
    }

    // Retrieve web content.
    private void fetchPage(String url) {
       // Perform basic validation (set characters to lowercase and add http:// or
https://).
        String lcase = url.toLowerCase();
        boolean validHeader = false;
        int i = 0;
        for (i = HTTP_PROTOCOL.length - 1; i >= 0; --i) {
            if ( -1 != lcase.indexOf(HTTP_PROTOCOL[i]) ) {
                validHeader = true;
                break;
            }
        }
        if ( !validHeader ) {
             // Prepend the protocol specifier if it is missing.
            url = HTTP_PROTOCOL[0] + url;
        }

        // Create a new thread for connection operations.
        _connectionThread.fetch(url);
    }
    // Display the content.
    private void updateContent(final String text) {
        /* This technique creates several short-lived objects but avoids
        * the threading issues involved in creating a static Runnable and
        * setting the text.
        */
        UiApplication.getUiApplication().invokeLater(new Runnable() {
            public void run() {
                _content.setText(text);
            }
        });
    }
    // Close the connection thread when the user closes the application.
    protected void onExit() {
        _connectionThread.stop();
    }
```

```
}
```

Using HTTPS connections

Note: The BlackBerry Internet Service Browser does not allow Java applications to initiate HTTP, HTTPS, or TCP connections.

Open an HTTPS connection

To open an HTTPS connection, invoke `Connector.open()`, specifying `https` as the protocol. Cast the returned object as an `HttpsConnection`.

```
HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/");
```

Specify proxy or end-to-end mode

By default, the connection uses HTTPS in proxy mode. Users can also set a handheld option to use end-to-end mode by default. See " HTTPS support" on page 151 for more information.

To open an HTTPS connection in end-to-end mode, add one of the following parameters to the connection string passed to `Connector.open()`:

Parameter	Description
END_TO_END_REQUIRED	This parameter specifies that an end-to-end HTTPS connection *must* be used from the handheld to the target server. If an end-to-end HTTPS connection cannot be set up, the connection is closed.
END_TO_END_DESIRED	This parameter specifies that an end-to-end HTTPS connection *should* be used from the handheld to the target server, if the handheld supports it. If the handheld does not support end-to-end TLS, and the user permits proxy TLS connections, then a proxy connection is used.

```
HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/
   ;END_TO_END_DESIRED");
```

Note: The modules for end-to-end HTTPS are not installed on handhelds by default; however, they are included with the BlackBerry Desktop Software version 3.6.0 or later. To load the required modules when the application is loaded onto the handheld, add the following tags to the .alx file for your application:

```
<requires id="net.rim.blackberry.crypto1"/>
<requires id="net.rim.blackberry.crypto2"/>
```

See " .alx files" on page 145 for more information.

Using socket connections

Note: The BlackBerry Internet Service Browser does not allow Java applications to initiate HTTP, HTTPS, or TCP connections.

Specifying TCP settings

Applications can set up a Transmission Control Protocol (TCP) socket connection, or an HTTP connection over TCP, in one of two modes:

- Proxy mode: The Mobile Data Service feature of the BlackBerry Enterprise Server sets up the TCP connection to the web server on behalf of the handheld.
- Direct mode: The handheld sets up a direct TCP connection to the web server.

Note: Using direct TCP mode requires that you work closely with service providers. Contact your service provider to verify that direct TCP socket connections are supported.

To specify TCP settings programmatically, add the optional `deviceside` parameter to the connection string passed to `Connector.open()`.

To specify TCP settings in handhelds on GSM networks, users click **TCP** on the handheld options.

Note: TCP only appears under the handheld options if the IT policy settings allow TCP connections.

If TCP settings are not specified, the following defaults are used.

Network	Default TCP setting	Alternate TCP setting
GSM	proxy mode	direct mode
iDEN	direct mode	proxy mode

See `Connector` in the *API Reference* for more information.

Open a socket connection

To open a socket connection, invoke `Connector.open()`, specifying `socket` as the protocol.

```
private static String URL = "socket://localhost:4444";
StreamConnection conn = null;
conn = (StreamConnection)Connector.open(URL);
```

Send and receive data on a socket connection

To send and receive data on a socket connection, acquire input and output streams using `openInputStream()` and `openOutputStream()`.

```
OutputStreamWriter _out = new OutputStreamWriter(conn.openOutputStream());
String data = "This is a test";
int length = data.length();
_out.write(data, 0, length);
InputStreamReader _in = new InputStreamReader(conn.openInputStream());
char[] input = new char[length];
for ( int i = 0; i < length; ++i ) {
   input[i] = (char)_in.read();
}
```

Close the connection

To close the input and output streams, and the socket connection, invoke `close()`.

Note: Each of the `close()` methods throws an `IOException`. You should implement error handling.

```
_in.close();
```

```
_out.close();
conn.close();
```

Using port connections

Using a serial or USB connection, handheld applications can communicate with desktop applications when they are connected to a computer using a serial or USB port. This type of connection can also be used to communicate with a peripheral device that plugs into the serial or USB port.

Note: If you are using the port connection to communicate with a desktop application, you must not have any other applications running that are using the serial or USB port, such as the BlackBerry Desktop Manager.

Open a port connection

To open a USB or serial port connection, invoke `Connector.open()`, specifying `comm` as the protocol and `COM1` or `USB` as the port.

```
private StreamConnection _conn = (StreamConnection)Connector.open(
"comm:COM1;baudrate=9600;bitsperchar=8;parity=none;stopbits=1");
```

Send data on a port connection

To send data on a port connection, acquire an output stream by invoking `openDataOutputStream()` or `openOutputStream()`.

```
DataOutputStream _dout = _conn.openDataOutputStream();
```

Use the write methods on the output stream to write data.

```
private String data = "This is a test";
_dout.writeChars(test);
```

Receive data on a port connection

To receive data on a port connection, acquire an input stream by invoking `openInputStream()` or `openDataInputStream()`.

```
DataInputStream _din = _conn.openInputStream();
```

Use the read methods on the input stream to read data.

```
String contents = _din.readUTF();
```

Note: You cannot read from the input stream on the main event thread, because this operation blocks until data is received. Create a separate thread on which to receive data.

Close a port connection

To close the input and output streams, and port connection, invoke `close()`.

Note: Each of the `close()` methods can throw an `IOException`. You should implement error handling.

```
_din.close();
_dout.close();
conn.close();
```

Using Bluetooth serial port connections

The Bluetooth API (`net.rim.device.api.bluetooth`) enables handheld applications to initiate a server or client Bluetooth serial port connection to a computer or other Bluetooth wireless technology enabled device.

Notes: Check for a `ControlledAccessException` when your application first accesses the Bluetooth API. This runtime exception is thrown if the system administrator restricts access to the Bluetooth API using application control. See " Application control" on page 12 for more information.

The handheld simulator does not support Bluetooth.

Open a Bluetooth serial port connection

To open a Bluetooth serial port connection, invoke `Connector.open()`, providing the serial port information returned by `BluetoothSerialPort.getSerialPortInfo()` as a parameter. The connection string returned by this method specifies `btspp://` as the protocol and one of the following items:

- If you are opening the connection as a client, the connection string returned by `getSerialPortInfo().toString()` contains the device id and port number on which the server device is listening.
- If you are opening the connection as a server, the connection string returned by `getSerialPortInfo().toString()` contains the port on which your device is listening.

```
BluetoothSerialPortInfo[] info = BluetoothSerialPort.getSerialPortInfo();
StreamConnection _conn = (StreamConnection)Connector.open( info.toString(),
   Connector.READ_WRITE );
```

Send and receive data

The procedures for sending and receiving data on a Bluetooth serial port connection are identical to those for any port connection. See “Send data on a port connection” on page 89 and “Receive data on a port connection” on page 89 for more information.

Close a port connection

To close the input and output streams, and the Bluetooth serial port connection, invoke `close()`.

```
if (_bluetoothConnection != null) {
   try {
      _bluetoothConnection.close();
   } catch(IOException ioe) {
   }
}
if (_din != null) {
   try {
        _din.close();
    } catch(IOException ioe) {
    }
}
if (_dout != null) {
    try {
        _dout.close();
    } catch(IOException ioe) {
    }
}
_bluetoothConnection = null;
_din = null;
_dout = null;
```

Code example

The following code sample is the client side of a simple Bluetooth serial port application. This application listens for data on the serial port and renders the data when it arrives.

Example: BluetoothSerialPortDemo.java

```
/**
 * BluetoothSerialPortDemo.java
 * Copyright (C) 2004 Research In Motion Limited.
 */

 /* The client side of a simple serial port demonstration app.
  * This application listens for text on the serial port and
  * renders the data when it arrives.
  */

package com.rim.samples.docs.bluetoothserialportdemo;

import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.bluetooth.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.baseapp.*;


public class BluetoothSerialPortDemo extends BaseApp implements
    BluetoothSerialPortDemoResResource {
    //statics ------------------------------------------------------------------
    private static ResourceBundle _resources;

    private static final int INSERT = 1;
    private static final int REMOVE = 2;
    private static final int CHANGE = 3;
    private static final int JUST_OPEN = 4;
 private static final int CONTENTS = 5;
 private static final int NO_CONTENTS = 6;

 static {
     _resources = 
ResourceBundle.getBundle(BluetoothSerialPortDemoResResource.BUNDLE_ID, 
BluetoothSerialPortDemoResResource.BUNDLE_NAME);
 }

 private EditField _infoField;
 private StreamConnection _bluetoothConnection;
 private DataInputStream _din;
 private DataOutputStream _dout;

 public static void main(String[] args)
 {
     BluetoothSerialPortDemo theApp = new BluetoothSerialPortDemo();
     theApp.enterEventDispatcher();
 }

 //constructor -----------------------------------------------------------
 public BluetoothSerialPortDemo()
 {
     MainScreen mainScreen = new MainScreen();
     mainScreen.setTitle(new LabelField(_resources.getString(TITLE), 
LabelField.USE_ALL_WIDTH));

     _infoField = new EditField(Field.READONLY);
     mainScreen.add(_infoField);

     mainScreen.addKeyListener(this);
     mainScreen.addTrackwheelListener(this);

     pushScreen(mainScreen);

     invokeLater(new Runnable() {
         public void run() {
             openPort();
         }
     });
 }

 protected void onExit() {
     closePort();
 }

 // Close the serial port.
 private void closePort() {
     if (_bluetoothConnection != null) {
         try {
             _bluetoothConnection.close();
         } catch(IOException ioe) {
         }
     }
     if (_din != null) {
         try {
                _din.close();
            } catch(IOException ioe) {
            }
        }
        if (_dout != null) {
            try {
                _dout.close();
            } catch(IOException ioe) {
            }
        }
        _bluetoothConnection = null;
        _din = null;
        _dout = null;
    }

    // Open the serial port.
    private void openPort() {
        try {
            if (_bluetoothConnection != null) {
                closePort();
            }
            BluetoothSerialPortInfo[] info =
BluetoothSerialPort.getSerialPortInfo();
            if( info == null || info.length == 0 ) {
               Dialog.alert( "No bluetooth serial ports available for connection."
);
                onExit();
                System.exit( 1 );
            }

            _bluetoothConnection = (StreamConnection)Connector.open(
info[0].toString(), Connector.READ_WRITE );
            _din = _bluetoothConnection.openDataInputStream();
            _dout = _bluetoothConnection.openDataOutputStream();
            new InputThread().start();

        } catch(IOException e) {
            invokeLater( new Runnable() {
                public void run() {
                    Dialog.alert("Unable to open serial port");
                    onExit();
                    System.exit(1);
                }
            });

        }
    }

    private class InputThread extends Thread {

        public void run() {
            try {
                int type, offset, count;
                String value;
                _dout.writeInt(JUST_OPEN);
                _dout.flush();
            for (;;) {
                type = _din.readInt();
                if (type == INSERT) {
                    offset = _din.readInt();
                    value = _din.readUTF();
                    insert(value, offset);
                } else if (type == REMOVE) {
                    offset = _din.readInt();
                    count = _din.readInt();
                    remove(offset, count);
                } else if (type == JUST_OPEN) {
                    // Send contents to desktop.
                    value = _infoField.getText();
                    if (value == null || value.equals("")) {
                        _dout.writeInt(NO_CONTENTS);
                        _dout.flush();
                    } else {
                        _dout.writeInt(CONTENTS);
                        _dout.writeUTF(_infoField.getText());
                        _dout.flush();
                    }
                } else if (type == CONTENTS) {
                    String contents = _din.readUTF();
                    synchronized(Application.getEventLock()) {
                        _infoField.setText(contents);
                    }
                } else if (type == NO_CONTENTS) {
                } else {
                    throw new RuntimeException();
                }
            }
        } catch(IOException ioe) {
            invokeLater(new Runnable() {
                public void run() {
                    Dialog.alert("Problems reading from or writing to serial
port.");
                    onExit();
                    System.exit(1);
                }
            });
        }
    }

}

private void insert(final String msg, final int offset) {
    invokeLater(new Runnable() {
        public void run() {
            _infoField.setCursorPosition(offset);
            _infoField.insert(msg);
        }
    });
}

private void remove(final int offset, final int count) {
    invokeLater(new Runnable() {
            public void run() {
                _infoField.setCursorPosition(offset+count);
                _infoField.backspace(count);
            }
        });
    }

    /**
     * Override makeMenu to add custom menu items.
     */
    protected void makeMenu(Menu menu, int instance)
    {
        if (_infoField.getTextLength() > 0) {
            menu.add(new MenuItem(_resources, MENUITEM_COPY, 100000, 10) {
                             public void run() {
                               Clipboard.getClipboard().put(_infoField.getText());
                             }
                         });
        }
        super.makeMenu(menu, instance);
    }
}
```

Using datagram connections

Datagram connections

BlackBerry handhelds support datagram connections using the User Datagram Protocol (UDP). Applications use UDP to communicate with standard network services.

Datagrams are independent packets of data that applications send over networks. A `Datagram` object is a wrapper for the array of bytes that is the payload of the datagram. To retrieve a reference to this byte array, invoke `getData()`.

Unlike HTTP connections, datagram connections are stateless: packets arrive in any order, and delivery is not guaranteed. Applications are responsible for making sure that the data payload of request datagrams is formatted according to the standards of the network service over which the datagram is transmitted. Applications must also be able to parse the datagrams that are sent back from the server.

Use datagram connections to send SMS messages. See " Sending and receiving SMS messages" on page 105 for more information.

Using UDP connections

To use a UDP connection, you must have your own infrastructure to connect to the wireless network, including an access point name (APN) for General Packet Radio Service (GPRS) networks.

Note: Datagram connections do not use the BlackBerry Infrastructure, so communication is not encrypted.

The APN of the simulator is `net.rim.gprs`.

The `javax.microedition.io.DatagramConnection` interface, which extends the `Connection` class, defines connections that send and receive datagrams. The `Datagram` interface defines the packets that are sent and received over a datagram connection.

Note: Using UDP connections requires that you work closely with service providers. Contact your service provider to verify that UDP connections are supported.

You might not be able to set up a UDP connection if your service provider does not support multiple PDP contexts. One PDP context is reserved for the blackberry.net APN, which is used for email. You might, however, be able to use blackberry.net as the APN for UDP. Contact your service provider for more information.

Open a UDP connection

To retrieve a `DatagramConnection`, invoke `Connector.open()` using the following format. Specify `udp` as the protocol.

```
(DatagramConnection)Connector.open("udp://host:dest_port[;src_port]/apn");
```

Parameter	Description
`host`	This parameter specifies the host address in dotted ASCII-decimal format.
`dest_port`	This parameter specifies the destination port at the host address (optional for receiving messages).
`src_port`	This parameter specifies the local source port (optional).
`apn`	This parameter specifies the network APN in string format.

Note: You can send and receive UDP datagrams on the same port.

To send data on a UDP connection, specify a destination port in the connection string. To receive data on a UDP connection, specify a source port in the connection string. To receive datagrams from all ports at the specified host, omit the destination port.

Note: To open a connection on a non-GPRS network, do not specify an APN. Include the slash mark after the source port. For example, the address for a CDMA network connection would be `udp://121.0.0.0:2332;6343/`.

Create a datagram

Invoke `DatagramConnection.newDatagram()`.

```
Datagram outDatagram = conn.newDatagram(buf, buf.length);
```

Add data to a datagram

Invoke `Datagram.setData()`. The format of the data is determined by the service that receives it.

```
byte[] buf = new byte[256];
outDatagram.setData(buf, buf.length);
```

Send data on a UDP connection

Invoke `send()` on a `DatagramConnection` instance.

```
conn.send(outDatagram);
```

Note: If an application attempts to send a datagram on a UDP connection and the recipient is not listening on the specified source port, an `IOException` is thrown.

Receive data on a UDP connection

Call `receive()` on the datagram connection.

```
byte[] buf = new byte[256];
Datagram inDatagram = conn.newDatagram(buf, buf.length);
conn.receive(inDatagram);
```

Note: The `receive()` method blocks other operations until it receives a packet. If the packet is lost, the application waits indefinitely. Set a timer to retransmit the request or close the connection if a reply does not arrive.

Extract data from a datagram

Invoke `getData()`. If you know the type of data that you are receiving, convert the data to the appropriate format.

```
String received = new String(inDatagram.getData());
```

Close the UDP connection

As with all connections in the MIDP framework, invoke `close()` on input and output streams to close them, and then invoke `close()` on the connection to close it.

Using Mobitex networks

The `DatagramConnectionBase` class provides methods that handle BlackBerry datagram connection and transmission operations over Mobitex networks.

Open a Mobitex datagram connection

Open a `DatagramConnection` using `Connector.open()`, and then cast it as a `DatagramConnectionBase`. The `DatagramConnectionBase` class implements `DatagramConnection` but provides additional methods that are necessary to register a datagram status listener.

To open a `DatagramConnection`, invoke `Connector.open()` and provide as a parameter a connection string using the following format:

`mobitex:<type>:<MAN>`

Parameter	Description
<*type*>	accepts the following values: " TEXT", " DATA", " STATUS" or " HPDATA{HPID}" (in which case HPID is in ASCII-decimal format)
<*MAN*>	Mobitex Access Number; accepts ASCII-decimal format

Note: If you open a server connection (a listener), leave the MAN blank.

```
// datagram connection <type> is DATA and the MAN is left blank for an incoming
// connection
DatagramConnection dc = (DatagramConnection)Connector.open("mobitex:DATA:");
DatagramConnectionBase dcb = (DatagramConnectionBase)dc;
```

Listening for datagram status events

If you want to register a datagram status listener, use a `DatagramBase` rather than a `Datagram` to hold data. `DatagramBase` implements the `Datagram` interface, but provides additional methods that are necessary to register a datagram status listener.

```
// dc is a DatagramConnection
Datagram d = dc.newDatagram(dc.getMaximumLength());
d.setAddress(address);
d.setData(raw, 0, raw.length);
DatagramBase db = (DatagramBase)d; //an error if this fails
```

Register a datagram status listener

To listen for events, such as the receipt of a datagram, implement the `DatagramStatusListener` interface. See `DatagramStatusListener` in the *API Reference* for a complete list of datagram status events.

To allocate a datagram ID and assign it to the `DatagramStatusListener` implicitly, invoke `DatagramConnectionBase.allocateDatagramId()`.

```
int id = dcb.allocateDatagramId(d);
```

Preallocating the datagram ID in this way ensures that your listener code is aware of the datagram that is associated with the ID.

Obtain datagram information

The `MobitexAddress` class encapsulates Mobitex addressing information, such as the Mobitex Access Number (MAN), the type of message, and the message status.

The `MobitexPacketHeader` class provides low-level access to the radio header field. To use the `MobitexPacketHeader` for all addressing operations and ignore the other `MobitexAddress` fields, invoke `MobitexAddress.setPacketHeader()`.

Obtain radio and network information

The `MobitexInfo` class provides objects to store general information about the state of the radio. The `MobitexInfo.MobitexCellInfo` class provides objects to store Mobitex cell information.

Code example

The `MobitexDemo.java` code example demonstrates the use of the Mobitex radio layer APIs.

Example: MobitexDemo.java

```
/*
 * MobitexDemo.java
 *
 * © Research In Motion Limited, 2003-2003
 * Confidential and proprietary.
 */

package com.rim.docs.samples.mobitexdemo;

import javax.microedition.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import java.util.*;
import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import java.io.*;
import com.rim.docs.samples.baseapp.*;

/**
 * A simple mobitex layer sample
 */
public final class MobitexDemo extends BaseApp implements MobitexDemoResource {
 private MainScreen _mainScreen;
 private EditField _pin;
 private EditField _data;
 private RichTextField _messages;
 private SenderThread _sendThread;
 private ReceiverThread _receiverThread;

 //statics ------------------------------------------------------------
 private static ResourceBundle _resources = 
ResourceBundle.getBundle(MobitexDemoResource.BUNDLE_ID, 
MobitexDemoResource.BUNDLE_NAME);

 static public void main(String[] args)
 {
     new MobitexDemo().enterEventDispatcher();
 }

 //menuitems ----------------------------------------------------------
 //cache the send menu item for reuse
 private MenuItem _sendMenuItem = new MenuItem(_resources, 
MOBITEXDEMO_MENUITEM_SEND, 100, 10) {
     public void run()
     {
         //don't execute on a blank address
         String pin = _pin.getText();
         String data = _data.getText();
         if ( pin.length() > 0  )
         {
             send(pin, data);
         }
     }
 };
 //cache the clear messages menu item for reuse
 private MenuItem _clearMessages = new MenuItem(_resources, 
MOBITEXDEMO_MENUITEM_CLEARMESSAGES, 105, 10) {
     public void run()
     {
         _messages.setText("");
     }
 };

 public MobitexDemo()
 {
     _mainScreen = new MainScreen();
     _mainScreen.setTitle( new 
LabelField(_resources.getString(MOBITEXDEMO_TITLE), LabelField.ELLIPSIS | 
LabelField.USE_ALL_WIDTH));

     _pin = new EditField(_resources.getString(MOBITEXDEMO_LABEL_PIN), null, 
Integer.MAX_VALUE, EditField.FILTER_PIN_ADDRESS);
     _mainScreen.add(_pin);

     _data = new EditField(_resources.getString(MOBITEXDEMO_LABEL_DATA), null);
     _mainScreen.add(_data);

     _mainScreen.add(new SeparatorField());
     _messages = new
RichTextField(_resources.getString(MOBITEXDEMO_CONTENT_DEFAULT));
     _mainScreen.add(_messages);

     _mainScreen.addKeyListener(this); //implemented by the super
     _mainScreen.addTrackwheelListener(this); //implemented by the super

     //start the helper threads
     _sendThread = new SenderThread();
     _sendThread.start();

     _receiverThread = new ReceiverThread();
     _receiverThread.start();

     pushScreen(_mainScreen); //push the main screen - a method on the
UiApplication class
 }

 //methods ------------------------------------------------------------
 /*public boolean keyChar(char key, int status, int time)
 {
     if (
UiApplication.getUiApplication().getActiveScreen().getLeafFieldWithFocus() ==
_pin && key == Characters.ENTER )
     {
         _sendMenuItem.run();
         return true; //I've absorbed this event, so return true
     }
     else
     {
         return super.keyChar(key, status, time);
     }
 }*/

 protected void makeMenu(Menu menu, int instance)
 {
     menu.add(_sendMenuItem);
     menu.add(_clearMessages);

     menu.addSeparator();

     super.makeMenu(menu, instance);
 }

 private void send(String pin, String data)
 {
     _sendThread.send(pin, data);
 }

 private void message(String msg)
 {
     System.out.println(msg);
     _messages.setText(_messages.getText() + msg + "\n");
 }
```

```
//innerclasses ------------------------------------------------------------
private class ReceiverThread extends Thread
{
    private DatagramConnection _dc;

    //shutdown the thread
    public void stop()
    {
        try {
            _dc.close();
        } catch(IOException e) {
            MobitexDemo.this.message(e.toString());
        }
    }

    public void run()
    {
        try {
            //inbound data connection - leave the MAN blank
            _dc = (DatagramConnection)Connector.open("mobitex:DATA:");
            for(;;)
            {
                Datagram d = _dc.newDatagram(_dc.getMaximumLength());
                _dc.receive(d);

                DatagramBase db = (DatagramBase)d;

                MobitexAddress ma = (MobitexAddress)db.getAddressBase();
                MobitexPacketHeader mph = ma.getPacketHeader();
                StringBuffer sb = new StringBuffer();
                sb.append("Recieved packet");
                sb.append("\nFROM:");
                sb.append(mph.getSourceAddress());
                sb.append("\nTO:");
                sb.append(mph.getDestinationAddress());
                sb.append("\nFLAGS:");
                sb.append(Integer.toHexString(mph.getPacketFlags()));
                sb.append("\nDATA:");
                sb.append(new String(db.getData()));

                MobitexDemo.this.message(sb.toString());
            }
        } catch (IOException e) {
            MobitexDemo.this.message(e.toString());
        }
    }
}

/**
 * The ConnectionThread class manages the datagram connection
 */
private class SenderThread extends Thread implements DatagramStatusListener
{
    private static final int TIMEOUT = 500; //ms
private Vector _sendQueue = new Vector(5);

private volatile boolean _start = false;
private volatile boolean _stop = false;

//queue something for sending
public void send(String pin, String data)
{
    synchronized(_sendQueue)
    {
        _sendQueue.addElement(new String[] { pin, data });
        _start = true;
    }
}

//shutdown the thread
public void stop()
{
    _stop = true;
}

public void run()
{
    for(;;)
    {
        String pin = null;
        String data = null;
        //Thread control\
        synchronized(_sendQueue)
        {
            while( !_start && !_stop)
            {
                //sleep for a bit so we don't spin
                try {
                    _sendQueue.wait(TIMEOUT);
                } catch (InterruptedException e) {
                    System.err.println(e.toString());
                }
            }
            if (_start)
            {
                String[] a = (String[])_sendQueue.firstElement();
                if ( a != null )
                {
                    pin = a[0];
                    data = a[1];
                }
                _start = false;
            }
            //exit condition
            else if ( _stop )
            {
                return;
            }
        }
```

```
                //open the connection and extract the data
                DatagramConnection dc = null;
                try {
                    String address = "DATA:" + pin;
                    dc = (DatagramConnection)Connector.open("mobitex:" + address);
                    DatagramConnectionBase dcb = (DatagramConnectionBase)dc; //an 
error if this fails

                    Datagram d = dc.newDatagram(dc.getMaximumLength());
                    byte[] raw = data.getBytes();
                    d.setAddress(address);
                    d.setData(raw, 0, raw.length);
                    DatagramBase db = (DatagramBase)d; //an error if this fails
                    dcb.allocateDatagramId(d); //allocate a datagram id - not 
necssary, but if we want to know about status
                    // for this particular datagram, then we can allocate the id 
here and log it for later follow up

                    //setup a status listener
                    db.setDatagramStatusListener(this);

                    dcb.send(d);

                    dc.close();

                } catch (IOException e) {
                    MobitexDemo.this.message(e.toString());
                }
                //we're done one connection so reset the start state
                _start = false;
            }
        }

        public void updateDatagramStatus(int dgId, int code, Object context)
        {
            String msg = "Datagram: " + Integer.toHexString(dgId) + "\nStatus: " + 
DatagramStatusListenerUtil.getStatusMessage(code);
            MobitexDemo.this.message(msg);
        }
    }

    protected void onExit()
    {
        _sendThread.stop();
        _receiverThread.stop();
    }
}
```

Sending and receiving SMS messages

Send and receive SMS messages in datagram packets using UDP. SMS datagram packets, which include the BlackBerry header information, have a fixed size of 160 bytes.

Note: SMS messaging is not fully supported on all networks. Check with your service provider to verify that the relevant networks have full or partial support for SMS messaging. In most cases, SMS is supported on GPRS and CDMA network-enabled handhelds.

If the service provider supports SMS, system administrators can also use an IT policy to control the use of SMS messaging by corporate users. Administrators can set the ENABLE_SMS item to TRUE or FALSE. The default is TRUE (SMS messaging is allowed).

Sending SMS messages

Open a datagram connection for sending SMS messages

Invoke `Connector.open()`. Provide a connection string using the following format, where *<peer_address>* is the phone number—Mobile Station ISDN Number (MSISDN)—of the recipient.

```
DatagramConnection _dc = Connector.open("sms://<peer_address>");
```

You can also omit the `peer_address` and invoke `Datagram.setAddress()` instead to set the destination address of the message.

Create an SMS message

Invoke `DatagramConnection.newDatagram()`.

```
Datagram smsMessage = conn.newDatagram(buf, buf.length);
```

Set SMS message contents

Invoke `setData()`.

```
private String _msg = "This is a test message";
byte[] data = _msg.getBytes();
smsMessage.setData(data, 0, data.length);
```

Send an SMS message

Note: Open network connections on a separate thread from the main application thread so the UI does not stall.

If you did not specify `peer_address` in the connection string, invoke `Datagram.setAddress()` to set the SMS address. To send the SMS message, invoke `DatagramConnection.send()`.

```
smsMessage.setAddress("sms://+15555551234");
_dc.send(datagram);
```

Code example

The SendSms.java code example demonstrates how to send an SMS message on a separate thread.

Example: SendSms.java

```
/**
 * SendSms.java
 * Copyright (C) 2002-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.smsdemo;

import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;

public class SendSms extends Application {
    private static final int MAX_PHONE_NUMBER_LENGTH = 32;
    // Members.
    private String addr = "15195551234";
    private String msg = "This is a test message.";
    private DatagramConnection _dc = null;
    private static String _openString = "sms://";
    public static void main(String[] args)  {
        new SendSms().enterEventDispatcher();
    }
    public SendSms() {
        try {
            _dc = (DatagramConnection)Connector.open(_openString);
            byte[] data = msg.getBytes();
            Datagram d = _dc.newDatagram(_dc.getMaximumLength());
            d.setAddress("//" + addr);
            _dc.send(d);
        } catch ( IOException e) {
        }
        System.exit(0);
    }
}
```

Receiving SMS message

Create a separate listener thread

Listen for messages on a separate thread from the main application thread so that the UI does not stall.

Open a datagram connection

Invoke `Connector.open()`. Provide a connection string using the following format:

`_dc =  (DatagramConnection)Connector.open("sms://<peer_address><port>");`

where:

- *<peer_address>* is the phone number—Mobile Station ISDN Number (MSISDN)—of the receiver
- *<port>* is the port number for which the application receives SMS messages

Retrieve a datagram

Create a `Datagram` object to store the datagram. To retrieve the SMS message datagram, invoke `receive()` on the datagram connection. This operation blocks until data is received.

```
Datagram d = _dc.newDatagram(160); // SMS messages have a fixed size of 160 bytes
_dc.receive(d);
```

Extract data from a datagram

To extract the address from the SMS message, invoke `Datagram.getAddress()`. To extract the data from the SMS message, invoke `Datagram.getData()`.

```
String address = d.getAddress();
String msg = new String(d.getData());
```

Code example

The ReceiveSms.java code example demonstrates how to receive an SMS message on a separate thread.

Example: ReceiveSms.java

```
/**
 * ReceiveSms.java
 * Copyright (C) 2002-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.smsdemo;

import net.rim.device.api.io.*;
import net.rim.device.api.system.*;
import javax.microedition.io.*;
import java.util.*;
import java.io.*;

public class ReceiveSms extends Application {

    private ListeningThread _listener;

    // Additional code required for complete sample.

    public static void main(String[] args) {
        new ReceiveSms().enterEventDispatcher();
    }

    ReceiveSms() {
        _listener = new ListeningThread();
        _listener.start();
    }

    private static class ListeningThread extends Thread {
        private boolean _stop = false;
        private DatagramConnection _dc;
        public synchronized void stop() {
            _stop = true;
            try {
                _dc.close(); // Close the connection so the thread returns.
            } catch (IOException e) {
                System.err.println(e.toString());
            }
        }
        public void run() {
            try {
                _dc = (DatagramConnection)Connector.open("sms://");
                for(;;) {
                    if ( _stop ) {
                        return;
                    }
                    Datagram d = _dc.newDatagram(_dc.getMaximumLength());
                    _dc.receive(d);
                    String address = new String(d.getData());
                    String msg = new String(d.getData());
                    System.out.println("Message received: " + msg);
                    System.out.println("From: " + address);
                    System.exit(0);
                }
            } catch (IOException e) {
                System.err.println(e.toString());
            }
        }
    }
}
```

Localizing applications

Resource files

Design applications so that they can be localized (adapted to specific languages and regions) without coding changes. Instead of including textual elements in your source code, store text strings in separate resource files. In your source code, use unique identifiers to map to the appropriate resource.

Storing text strings in separate resource files has two benefits:

- Text translation is more efficient because all of the text strings for a given locale are stored in a single file, outside your source code.
- Applications can dynamically retrieve the appropriate text to display to the user, based on the user locale.

The BlackBerry JDE includes a built-in resource mechanism for creating string resources. The Localization API is included in the `net.rim.device.api.i18n` package.

Note: MIDP applications do not support localization.

The resources for a given locale are stored in a `ResourceBundle` object. A `ResourceBundleFamily` object contains a collection of `ResourceBundles`, which groups the resources for an application. The application can switch languages, depending on the user locale, without requiring new resource bundles.

The IDE compiles each resource bundle into a separate compiled .cod file. You can load the appropriate .cod files onto handhelds with the other .cod files for the application.

File required for localization	Description	Example
Resource header file	This file defines descriptive keys for each localized string. When the IDE builds a project, it creates a resource interface with the same name as the .rrh file and appends `Resource`. For example, if you create `AppName.rrh`, the interface is `AppNameResource`.	AppName.rrh
Resource content file (root locale)	This file maps resource keys to string values for the root (global) locale. It has the same name as the resource header file.	AppName.rrc
Resource content file (specific locales)	This file maps resource keys to string values for specific locales (language and country). Files have the same name as the resource header file, followed by an underscore (_) and the language code, and then, optionally, an underscore (_) and country code. Two-letter language and country codes are specified in ISO-639 and ISO-3166, respectively.	AppName_en.rrc AppName_en_GB.rrc AppName_fr.rrc
Initialization file	This file initializes the resource bundle mechanism. This file is required only when resources are compiled as a separate project.	init.java

Resource inheritance

Resources are organized in a hierarchy based on inheritance. If a string is not defined in a locale, a string from the next closest locale is used.

Adding localization support to applications

Add resource header files

1. In the IDE, on the **File** menu, click **New**.
2. In the **Source file name** field, type a file name.
3. Click **Browse**.
4. Select the folder that contains the file.
5. Click **OK**.
6. In the field, type the package name, for example, **com.rim.samples.docs.countryinfo**.
7. Click **OK**.
8. Click **Yes**.
9. Leave the file that appears in the text editor empty except for the package statement.
10. Add the file to your project by right-clicking the file in the right pane, and then clicking **Insert into project**.

Add resource content files

Create three resource content files in the same folder where `CountryInfo.java` is located: `CountryInfo.rrc` (root locale), `CountryInfo_en.rrc` (English), and `CountryInfo_fr.rrc` (French).

1. On the **File** menu, click **New**.
2. Type a file name and location.
3. Click **OK**.
4. Click **Yes**.
5. Leave the file empty.
6. Add the .rrc file to the application project by right-clicking the file in the right pane, and then clicking **Insert into project**.

Add resources

1. In the IDE, double-click a resource header file.

2. On the **Root** tab, type resource keys and values for each string or string array in your application.

   Each row defines a single resource. The **Keys** column displays a descriptive name for the resource. This is the name that you use in your code to retrieve the localized text. The **Values** column displays the text for this resource in a particular locale.

Tip: To add an array of values for a single resource key, in the resource editor, right-click a resource and click **Convert to Multiple Values**. Add one or more values to the array.

3. To specify a different text string in other locales, select the tab for a locale, such as **fr** for the French language.
4. In the **Value** cell for the resource, type the text string for the locale. If you do not define a value for a resource in a particular locale, the value for the root locale is used.

Tip: Type unicode characters directly into the **Value** cell. Visit http://www.unicode.org for more information.

Set an application title

You can provide a localized application title to display on the handheld Home screen. If you do not provide a resource for the application title, the value entered into the **Title** field on the **Application** tab of the project properties window is used.

1. In the resource editor, add a resource for the application title, such as `APPLICATION_TITLE`.
2. Type a value for this resource in each locale that you support.

Tip: To create a shortcut key for an application, add the unicode underscore character (\u0332) after the letter that you want to use as a shortcut key. A shortcut key is a key that a user can press on the Home screen to start the application.

3. In the IDE, right-click the application project, and then click **Properties**.
4. Click the **Resources** tab.
5. Select the **Title Resource Available** option.
6. From the **Resource Bundle** drop-down list, select the resource header file name to use for this application.
7. From the **Resource Id** drop-down list, select the resource to use for the application title, such as `APPLICATION_TITLE`.

Code example

The CountryInfo.java sample demonstrates how to store text strings in separate resource files for specific locales rather than providing text strings directly in the code. In your source code, you retrieve the string from the resource to display the appropriate text for the user locale.

Tip: The CountryInfo.java sample adds resources to the project for a single application. If you are creating a suite of applications, organize resources into separate projects for each locale. The sample workspace that is included in the JDE provides an example of this organization. See the *IDE Online Help* for more information.

Example: CountryInfo.java

```
/**
 * CountryInfo.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */
package com.rim.samples.docs.countryinfo;
```

```
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;

public class CountryInfo extends UiApplication {
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
    }

    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
    }
}

final class HelloWorldScreen extends MainScreen implements CountryInfoResource {
    private  InfoScreen  _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;

    private static ResourceBundle _resources = ResourceBundle.getBundle(
            BUNDLE_ID, BUNDLE_NAME);

    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE),
                LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new ObjectChoiceField(
                _resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
    }

    public boolean onClose() {
        Dialog.alert(_resources.getString(CLOSE));
        System.exit(0);
        return true;
    }

    private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {
        public void run() {
            select = choiceField.getSelectedIndex();
            _infoScreen = new InfoScreen();
            UiApplication.getUiApplication().pushScreen(_infoScreen);
        }
    };

    private MenuItem _closeItem = new MenuItem(_resources, MENUITEM_CLOSE,
            200000, 10) {
        public void run() {
            onClose();
        }
    };

    protected void makeMenu( Menu menu, int instance ) {
        menu.add(_viewItem);
        menu.add(_closeItem);
    }

    private class InfoScreen extends MainScreen {
        public InfoScreen() {
            super();
            LabelField lf = new LabelField();
            BasicEditField popField = new BasicEditField(
                    _resources.getString(FIELD_POP), null, 20, Field.READONLY);
            BasicEditField langField = new BasicEditField(
                    _resources.getString(FIELD_LANG), null, 20, Field.READONLY);
            BasicEditField citiesField = new BasicEditField(
                    _resources.getString(FIELD_CITIES), null, 50, Field.READONLY);
            add(lf);
            add(new SeparatorField());
            add(popField);
            add(langField);
            add(citiesField);
            if (select == 0) {
                lf.setText(_resources.getString(FIELD_US));
                popField.setText(_resources.getString(FIELD_US_POP));
                langField.setText(_resources.getString(FIELD_US_LANG));
                citiesField.setText(_resources.getString(FIELD_US_CITIES));
            } else if (select == 1) {
                lf.setText(_resources.getString(FIELD_CHINA));
                popField.setText(_resources.getString(FIELD_CHINA_POP));
                langField.setText(_resources.getString(FIELD_CHINA_LANG));
                citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
            } else if (select == 2) {
                lf.setText(_resources.getString(FIELD_GERMANY));
                popField.setText(_resources.getString(FIELD_GERMANY_POP));
                langField.setText(_resources.getString(FIELD_GERMANY_LANG));
                citiesField.setText(
                    _resources.getString(FIELD_GERMANY_CITIES));
            }
        }
    }
}
```

Retrieving strings from a resource file

Implement the resource interface

For internationalization, implement the appropriate resource interface. The IDE compiles this interface from the resource header (.rrh) automatically. The interface has the same name as the .rrh file, with `Resource` appended to it.

```
public class HelloWorldScreen extends MainScreen implements CountryInfoResource
{ ... }
```

Retrieve the resource bundle

Declare a class variable to hold the resource bundle for this application. A `ResourceBundle` object contains all localized resources, such as strings, for an application. An application can select the appropriate bundles at runtime based on its locale.

```
private static ResourceBundle _resources = ResourceBundle.getBundle(BUNDLE_ID,
   BUNDLE_NAME);
```

To retrieve the appropriate bundle family, invoke `getBundle()`. The IDE creates the `BUNDLE_ID` and `BUNDLE_NAME` constants when it creates the resource interface as part of building the project.

Create menu items using resources

To create `MenuItem` objects using resources, use the `MenuItem` constructor that accepts a resource bundle and a resource instead of a `String` for the name of the menu item. Do not implement `toString()`, because the text of the menu item is provided by the resource.

```
private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {
   public void run() {
      select = choiceField.getSelectedIndex();
      _infoScreen = new InfoScreen();
      UiApplication.getUiApplication().pushScreen(_infoScreen);
   }
};
```

Replace text strings with the appropriate resources

For each field that appears on the main screen, replace the text string with the appropriate resource. Invoke `getString()` or `getStringArray()` to retrieve the string for the appropriate language.

```
LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE),
   LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
add(new RichTextField(_resources.getString(FIELD_TITLE)));
String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
choiceField = new ObjectChoiceField(_resources.getString(FIELD_CHOICE), choices);
```

Code example

The following example modifies the `CountryInfo.java` sample to retrieve strings from a resource file.

Example: CountryInfo.java (with localization support)

```
/**
 * CountryInfo.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */
```

```
package com.rim.samples.docs.localization;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.system.*;
import net.rim.device.api.i18n.*;

public class CountryInfo extends UiApplication {
    public static void main(String[] args) {
        CountryInfo theApp = new CountryInfo();
        theApp.enterEventDispatcher();
    }
    public CountryInfo() {
        pushScreen(new HelloWorldScreen());
    }
}
final class HelloWorldScreen extends MainScreen implements CountryInfoResource {
    private  InfoScreen  _infoScreen;
    private ObjectChoiceField choiceField;
    private int select;
    private static ResourceBundle _resources = ResourceBundle.getBundle( BUNDLE_ID,
    BUNDLE_NAME );
    public HelloWorldScreen() {
        super();
        LabelField title = new LabelField(_resources.getString(APPLICATION_TITLE),
        LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField(_resources.getString(FIELD_TITLE)));
        String choices[] = _resources.getStringArray(FIELD_COUNTRIES);
        choiceField = new ObjectChoiceField(
        _resources.getString(FIELD_CHOICE), choices);
        add(choiceField);
    }
    public boolean onClose() {
        Dialog.alert(_resources.getString(CLOSE));
        System.exit(0);
        return true;
    }
    private MenuItem _viewItem = new MenuItem(_resources, MENUITEM_VIEW, 110, 10) {
        public void run() {
            select = choiceField.getSelectedIndex();
            _infoScreen = new InfoScreen();
            UiApplication.getUiApplication().pushScreen(_infoScreen);
        }
    };
    private MenuItem _closeItem = new MenuItem(_resources, MENUITEM_CLOSE, 200000,
    10) {
        public void run() {
            onClose();
        }
    };
    protected void makeMenu( Menu menu, int instance ) {
        menu.add(_viewItem);
        menu.add(_closeItem);
    }
    private class InfoScreen extends MainScreen {
        public InfoScreen() {
            super();
            LabelField lf = new LabelField();
            BasicEditField popField = new BasicEditField(
            _resources.getString(FIELD_POP), null, 20, Field.READONLY);
            BasicEditField langField = new BasicEditField(
            _resources.getString(FIELD_LANG), null, 20, Field.READONLY);
            BasicEditField citiesField = new BasicEditField(
            _resources.getString(FIELD_CITIES), null, 50, Field.READONLY);
            add(lf);
            add(new SeparatorField());
            add(popField);
            add(langField);
            add(citiesField);
            if (select == 0) {
                lf.setText(_resources.getString(FIELD_US));
                popField.setText(_resources.getString(FIELD_US_POP));
              langField.setText(_resources.getString(FIELD_US_LANG));
                citiesField.setText(_resources.getString(FIELD_US_CITIES));
            } else if (select == 1) {
                lf.setText(_resources.getString(FIELD_CHINA));
                popField.setText(_resources.getString(FIELD_CHINA_POP));
                langField.setText(_resources.getString(FIELD_CHINA_LANG));
                citiesField.setText(_resources.getString(FIELD_CHINA_CITIES));
            } else if (select == 2) {
                lf.setText(_resources.getString(FIELD_GERMANY));
                popField.setText(_resources.getString(FIELD_GERMANY_POP));
                langField.setText(_resources.getString(FIELD_GERMANY_LANG));
                citiesField.setText(
                _resources.getString(FIELD_GERMANY_CITIES));
            }
        }
    }
}
```

Managing resource files for application suites

To set up projects in the IDE that group resource files for each locale, perform the following actions:

1. Create resource projects.
2. Specify output file names.
3. Create an initialization file.
4. Add files to appropriate resource projects.

Create resource projects

Create a project for each resource bundle (locale), including the root locale.

Give the projects for each locale the same name as the project for the root locale, followed by a double underscore (__), the language code, and, optionally, an underscore (_) followed by the country code. For example, if the root locale project is named `com_company_app`, the projects for each locale would be named `com_company_app__en`, `com_company_app__en_GB`, `com_company_app__fr`.

Specify output file names

1. Right-click the project, and then click **Properties**.
2. Click the **Build** tab.
3. In the **Output file name** field, type a name for the compiled file, without a file name extension.

Note: The output file names for all resource locale projects must be the same as for the root locale, followed by a double underscore and the appropriate language and country codes. For example, if the output file name for the root locale project is `com_company_app`, the output file name for the French-language locale must be `com_company_app__fr`.

Create an initialization file

When you compile resources in a separate project, create an initialization file (for example, `init.java`). The IDE provides a built-in initialization mechanism, so that you only need to create an empty initialization class with an empty `main()`.

```
package com.rim.samples.device.resource;
import net.rim.device.api.i18n.*;
public class init {
   public static void main (String[] args) { }
}
```

Add files to appropriate resource projects

Create one resource header file for each application and one resource content file for each application, for each supported locale. Organize the resource files into projects.

1. Add the initialization file (`init.java`) to each resource project. Do not include the initialization file in the application projects.
2. Add the resource header (.rrh) files to the projects for each application and also to each resource project. This is necessary to define the dependency between the application project and its resource projects.
3. In each resource project, right-click each .rrh file, and then click **Properties**.
4. Select **Dependency only. Do not build**.
5. Add the resource content (.rrc) files to the resource projects for the appropriate locales.

Tip: If you support a large number of locales, create a single library project for all resource header (.rrh) files and set the project type to **Library**. For each resource locale in this project, define a dependency between the projects.

Using IT policies

IT policies

The BlackBerry IT policy API (`net.rim.device.api.itpolicy`) enables applications to access the IT policy database on handhelds. Applications can retrieve custom IT policy settings to change their behavior or functionality accordingly.

Note: The IT policy API enables applications to retrieve values for custom (third-party) IT policy items only. Applications cannot retrieve values for standard IT policy items.

Each IT policy item consists of a descriptive key and a value. The value can be a string, integer, or Boolean value. For example, the `AllowPhone` policy can have a value of `true` or `false`.

With the BlackBerry Enterprise Server version 3.5 or later for Microsoft® Exchange and BlackBerry Handheld Software version 3.5 or later, handheld policy settings are synchronized and updated wirelessly. With earlier versions of handheld software, handheld policy settings are updated when the user synchronizes the handheld with the desktop.

See the *BlackBerry Enterprise Server for Microsoft Exchange Handheld Management Guide* for more information.

Retrieving IT policy items

To retrieve the value of an IT policy item, invoke `getString()`, `getInteger()`, or `getBoolean()`.

Retrieve custom policies

To retrieve custom third-party IT policies by name, use the form of each method that accepts a `String` parameter.

```
public static String getString( String name );
public static boolean getBoolean( String name, boolean defaultValue );
public static int getInteger( String name, int defaultValue );
```

The `defaultValue` parameter specifies the return value if the parameter has not been set.

Listening for policy changes

A global event is generated when the IT policy database is updated on the handheld.

To use IT policies, applications implement the `GlobalEventListener` interface and register this listener to receive global events. The `GlobalEventListener.eventOccurred()` method is invoked when a global event, such as a change in IT policies, occurs. In its implementation of `eventOccurred()`, applications can retrieve values for IT policy items that they use to determine whether values have changed.

Controlling application downloads

With the BlackBerry Enterprise Server version 3.6 or later for Microsoft Exchange or BlackBerry Enterprise Server version 2.2 or later for IBM Lotus® Domino®, system administrators can set the following IT policies to control the use of third-party applications.

IT Policy	Default	Description
DISABLETHIRDPARTYAPPLICATIONDOWNLOADS	FALSE	Determines whether the user can install third-party applications, either wirelessly or using the desktop software. Administrators can install approved applications for the user, and then set this IT policy to TRUE to prevent the user from installing additional applications.
ALLOWAPPRUN	TRUE	Determines whether third-party applications can run on the handheld. For example, set this IT policy to FALSE to prevent users from using third-party applications that they have already downloaded.

Additional IT policies enable the corporate IT administrator to control the access of third-party applications to handheld resources, such as the persistent store, and to the network.

See the *BlackBerry Enterprise Server Handheld Management Guide* for more information on setting IT policies.

Code example

The ITPolicyDemo.java sample implements IT policy controls.

Example: ITPolicyDemo.java

```
/**
 * ITPolicyDemo.java
 * Copyright (C) 2002-2004 Research In Motion Limited.
 */

package com.rim.samples.docs.itpolicy;


import net.rim.device.api.system.*;
import net.rim.device.api.itpolicy.*;

public class ITPolicyDemo extends Application implements GlobalEventListener {
    public static void main(String[] args) {
        ITPolicyDemo app = new ITPolicyDemo();
        app.enterEventDispatcher();
    }
    ITPolicyDemo() {
        this.addGlobalEventListener(this);
       boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled", true);
        System.out.println("App Enabled: " + appEnabled);
        System.exit(0);
    }
    public void eventOccurred(long guid, int data0, int data1, Object obj0, Object
   obj1) {
        if (guid == ITPolicy.GUID_IT_POLICY_CHANGED ) {
            String security = ITPolicy.getString("DemoSecurityLevel");
            boolean appEnabled = ITPolicy.getBoolean("DemoAppEnabled", true);
            int retries = ITPolicy.getInteger("DemoAppRetries", 10);
        }
    }
}
```

Creating client/ server push applications

Push applications

Note: Push applications require BlackBerry Enterprise Server version 3.5 or later for Microsoft Exchange, or BlackBerry Enterprise Server version 2.2 or later for IBM Lotus Domino, with the Mobile Data Service enabled.

Push applications enable you to send new web content and alerts to specific users. Users do not have to request or download the data because the push application delivers the information as it becomes available.

There are two types of push applications:

- Browser push applications: Web content is sent to the browser on the handheld. The BlackBerry Browser configuration supports Mobile Data Service push applications. The WAP Browser configuration supports WAP push applications. The Internet Browser configuration does not support push applications. See the *BlackBerry Browser Developer Guide* for information on writing a browser push application.
- Client/server push applications: Data is sent to a custom Java application on the handheld. Client/server push applications consist of a custom client application for the handheld and a server-side application that pushes content to it. This approach provides more control over the type of content that you can send out and how this data is processed and displayed on the handheld compared to browser push applications.

Client/ server push requests

Applications can push content to handhelds using one of two methods:

- Push Access Protocol (PAP), which is part of the WAP 2.0 specification
- RIM push

Note: The Mobile Data Service only queues 1000 push requests, including both RIM and PAP push requests. The Mobile Data Service responds to the server with an error if it receives more than 1000 requests.

Both push service implementations support the following tasks:

- sending a server-side push submission
- specifying a reliability mode for the push submission

    - In transport-level reliability mode, messages are considered to be delivered when the Mobile Data Service receives acknowledgements for all sent packets.
    - In application-level reliability mode, messages are considered to be delivered when the Mobile Data Service receives confirmation from the application that initiated the push.
- specifying a deliver-before time-stamp for the push submission
- requesting a result notification of the push submission

The PAP implementation supports the following additional tasks:

- specifying a deliver-after timestamp for the push submission
- cancelling a push request submission
- querying the status of a push request submission

PAP pushes are stored in a database, whereas RIM pushes are stored in RAM. RIM pushes may be lost if the server is rebooted.

Transcoding

If applicable, the Mobile Data Service applies a transcoder to push requests according to its transcoder rules.

Push requests can override these rules to request a specific transcoder by using the `transfer-encoding` header. For example, if the HTTP header `transfer-encoding: vnd.wap.wml` is set, the Mobile Data Service runs the `wml` transcoder before it pushes the data to the handheld.

Transport-level acknowledgement

When the push arrives at a handheld, the Mobile Data Service initiates a connection to the URL specified in the push request to inform the server of the delivery. Transport-level acknowledgement is available in the BlackBerry Handheld Software version 3.6 or later.

Application-level acknowledgement

When the push arrives at a handheld, the application acknowledges the content. The Mobile Data Service initiates a connection to the URL specified during the push request to inform the server of the delivery. If an error is encountered, the Mobile Data Service sends an error message to the server. Application-level acknowledgement is available in handheld software version 4.0.

RIM push provides an application-preferred option, which uses application-level acknowledgement in handheld software version 4.0 and transport-level acknowledgement otherwise.

Note: The Mobile Data Service version 4.0 or earlier cannot query the BlackBerry Enterprise Server for device characteristics. To obtain device characteristics, the Mobile Data Service must receive an HTTP request before it receives a push request.

To provide the necessary requests, browse to a web page using the BlackBerry Browser and a Mobile Data Service browser configuration.

Send a RIM push request

To push data to handhelds using RIM push, send an HTTP POST request using the following format, where <*destination*> is the destination PIN or email address, <*port*> is the destination port, <*uri*> is the URI sent to the device, and <*content*> is a byte stream:

```
/push?DESTINATION=<destination>&PORT=<port>&REQUESTURI=<uri><headers><content>
```

The following headers are valid for RIM push requests:

HTTP header	Description
`X-RIM-Push-ID`	This header specifies a unique message ID, which can be used to cancel or check the status of a message. Typically, specify a URL in combination with a value, such as such as 123@rim.com.
`X-RIM-Push-NotifyURL`	This header specifies the URL to which a result notification is sent.
`X-RIM-Push-Reliability-Mode`	This header specifies the delivery reliability mode of the content—transport-level (TRANSPORT), application-level (APPLICATION) and application preferred (APPLICATION-PREFERRED).
`X-RIM-Push-Deliver-Before`	This header specifies the date and time by which the content must be delivered to the handheld. Content that has not been delivered before this date is not delivered.
`X-RIM-Push-Priority`	This header specifies the priority of channel push messages. Permitted strings include none (default), low, medium, and high. If the priority is low, medium or high, users receive notification of channel updates. If the priority is high, a status dialog accompanies the notification.

Send a PAP push request

To push data to handhelds using PAP, send an HTTP POST request using the following format, where <*destination*> is the destination PIN or email address, and <*port*> is the destination port:

```
/push?DESTINATION=<destination>&PORT=<port>&REQUESTURI=/pap
```

The request is a MIME multipart message, which consists of the following items:

- an XML document specifying the control entity
- the push content

For example, the control entity might contain information for the handheld address, message ID, and delivery timestamps.

Use the PAP Document Type Definition (DTD) to specify the following attributes:

XML control entity attributes	Description	Example
`X-Wap-Application-Id`	This entity attribute specifies the equivalent of the REQUEST URI HTTP parameter for RIM push.	"/"
`push-id`	Specifies a unique message ID. Additionally, this control entity attribute can be used to cancel or check the status of a message. It is recommended that you use a URL in combination with a value. For example, 123@wapforum.org	123@wapforum.org
`ppg-notify-requested-to`	Specifies the URL that result notification is sent to.	http://wapforum:8080/ReceivePAPNotification
`deliver-before-timestamp`	Specifies the date and time by which the content must be delivered to the handheld. Content that has not been sent by this date is not delivered.	2004-01-20T22:35:00z
`deliver-after-timestamp`	Specifies the date and time after which content is delivered to the handheld. Content is not delivered before this date.	2004-01-20T21:35:00z
`address-value`	Specifies the address of the handheld that the push content is sent to. The *destination* is the destination email address or PIN.	WAPPUSH=*destination*%3A*port*/TYPE=USER@rim.net

XML control entity attributes	Description	Example
`delivery-method`	Specifies the delivery reliability mode of the content, transport-level or application-level.	confirmed; unconfirmed

See the *Push Access Protocol (WAP-247-PAP-20010429-a)* specification for more information on writing server-side push applications using PAP. See the PAP 2.0 DTD for information on the WAP Push DTDs.

Example: PAP push request

```
Content-Type: multipart/related; type="application/xml"; boundary=asdlfkjiurwghasf
X-Wap-Application-Id: /

--asdlfkjiurwghasf
Content-Type: application/xml

<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"  "http://www.wapforum.org/DTD/
   pap_2.0.dtd">
<pap><push-message   push-id="a_push_id" ppg-notify-requested-to="http://
   foo.rim.net/ReceiveNotify">
      <address address-value="WAPPUSH=john.doe%40acme.com%3A7874/
   TYPE=USER@rim.net"/>
      <quality-of-service delivery-method="unconfirmed"/>
</push-message></pap>

--asdlfkjiurwghasf
Content-Type: text/html

<html><body>Hello, PAP world!</body></html>
--asdlfkjiurwghasf--
```

Send a PAP push cancellation request

Use the following header to cancel a push submission that has already been sent to the Mobile Data Service.

XML control entity attributes	Description	Example
`cancel-message push-id`	Specifies cancelling the push message that was previously submitted. You must include the address attribute in this request.	123@wapforum.org

Example: PAP push cancellation request

```
Content-Type: application/xml

<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
   "http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
<cancel-message push-id=“a_push_id">
   <address address-value=
         “WAPPUSH=john.doe%40acme.com%3A7874/TYPE=USER@rim.net”/>
```

```
</cancel-message>
</pap>
```

Send a PAP push query request

To query the status of a push submission that has already been sent to the Mobile Data Service, use the following header.:

XML control entity attributes	Description	Example
`statusquery-message push-id`	Specifies the push message for which status is desired. A response is returned with one of the following message states: delivered, pending, undeliverable, expired, rejected, timeout, cancelled, aborted or unknown. You must include the address attribute in this request.	123@wapforum.org

Example: PAP push status query request

```
Content-Type: application/xml

<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
   "http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
<statusquery-message push-id="a_push_id">
   <address address-value=
         "WAPPUSH=john.doe%40acme.com%3A7874/TYPE=USER@rim.net"/>
</statusquery-message>
</pap>
```

Writing a client-side push application

Create a listening thread

Send and receive data on a separate thread so that you do not block the main event thread.

Open an input stream

Open the connection once and keep the connection open. Re-open the connection only if an IOException occurs. Do not close and re-open the connection every time pushed data is received, because pushed data can be lost if it arrives before `Connector.open()` is invoked again after a previous push.

To avoid conflicts with other applications, choose a high port number. Port numbers must be from 1 to 65535. Port 7874 is reserved for the BlackBerry Browser.

```
StreamConnectionNotifier _notify =
   (StreamConnectionNotifier)Connector.open("http://:6234");
// open a server-side socket connection
StreamConnection stream = _notify.acceptAndOpen();
// open an input stream for the connection
InputStream input = stream.openInputStream();
```

Close the stream connection notifier

Invoke `close()` on the stream connection notifier.

```
_notify.close();
```

Code example

The HTTPPushDemo.java sample demonstrates how to write a handheld application that listens for inbound data from a web server. You create a listening thread to listen for image data on a specific port and then display it when it arrives.

Example: HTTPPushDemo.java

```
/**
 * HTTPPushDemo.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.httppush;

import java.io.*;
import javax.microedition.io.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;
import net.rim.device.api.system.*;
import net.rim.device.api.util.*;
import com.rim.samples.docs.baseapp.*;

public class HTTPPushDemo extends BaseApp {
    // Constants.
    private static final String URL = "http://:6234";
    private static final int CHUNK_SIZE = 256;

    // Fields.
    private ListeningThread _listeningThread;
    private MainScreen _mainScreen;
    private RichTextField _infoField;
    private BitmapField _imageField;

    public static void main(String[] args) {
        HTTPPushDemo theApp = new HTTPPushDemo();
        theApp.enterEventDispatcher();
    }
```

```
/**
* Create a separate listening thread so that you do not
* block the application’s main event thread.
*/
private class ListeningThread extends Thread {
    private boolean _stop = false;
    private StreamConnectionNotifier _notify;

    public synchronized void stop() {
        _stop = true;
        try {
            _notify.close(); // Close the connection so thread returns.
        } catch (IOException e) {
            System.err.println(e.toString());
        } catch (NullPointerException e) {
            // The notify object likely failed to open, due to an IOException.
        }
    }
    public void run() {
        StreamConnection stream = null;
        InputStream input = null;
        try {
            synchronized(this) {
                // Open the connection once or re-open after an IOException.
                _notify = (StreamConnectionNotifier)Connector.open(URL);
            }
            while (!_stop) {
                // NOTE: This method blocks until data is received.
                stream = _notify.acceptAndOpen();
                input = stream.openInputStream();

                // Extract the data from the input stream.
                DataBuffer db = new DataBuffer();
                byte[] data = new byte[CHUNK_SIZE];
                int chunk = 0;
                while ( -1 != (chunk = input.read(data)) ) {
                    db.write(data, 0, chunk);
                }
                input.close();
                data = db.getArray();
                updateBitmap(data);
            }
        } catch (IOException e) {
            System.err.println(e.toString()); // It is likely the stream was
closed.
        }
    }
}
// Constructor.
public HTTPPushDemo() {
    _mainScreen = new MainScreen();
    _mainScreen.setTitle(new LabelField("Latest Logos",
LabelField.USE_ALL_WIDTH));
    _infoField = new RichTextField();
    _mainScreen.add(_infoField);
    _mainScreen.add(new SeparatorField());
```

```
            _imageField = new BitmapField(null, BitmapField.HCENTER|BitmapField.TOP);
            _mainScreen.add(_imageField);
            _mainScreen.addKeyListener(this);
            _mainScreen.addTrackwheelListener(this);

            _listeningThread = new ListeningThread();
            _listeningThread.start();

            _infoField.setText("Application is listening...");
            pushScreen(_mainScreen);
        }
        private void updateBitmap(final byte[] data) {
            Application.getApplication().invokeLater(new Runnable() {
                public void run() {
                    // Query the user to load the received image.
                    String[] choices = {"OK", "CANCEL"};
                    if ( 0 != Dialog.ask("Do you want to display latest logo?",
                            choices, 0) ) {
                        return;
                    }

                _infoField.setText("Image received. Size: "+ data.length);
               _imageField.setBitmap(Bitmap.createBitmapFromPNG(data, 0,data.length));
                }
            });
        }
        protected void onExit() {
            // Stop the listening thread.
            _listeningThread.stop();
            try {
                _listeningThread.join();
            } catch (InterruptedException e) {
                System.err.println(e.toString());
            }
        }
    }
```

Writing a server-side push application

Any programming language that can establish an HTTP connection can be used to create a push application. The following sections use standard Java to demonstrate a server-side push application.

Construct the push URL

Format RIM push requests as follows:

/push?DESTINATION=<*destination*>&PORT=<*port*>&REQUESTURI=<*uri*><*headers*><*content*>

See " Send a RIM push request" on page 122 for more information on constructing a URL for a RIM push request.

Format PAP push requests as follows:

```
/push?DESTINATION=<destination>&PORT=<port>&REQUESTURI=/pap
```

See " Send a PAP push request" on page 123 for more information on constructing a ULR for a PAP push request.

Connect to the BlackBerry Enterprise Server

Invoke `openConnection()` on the push URL, and then cast the returned object as an `HttpURLConnection`. An `HttpURLConnection` represents a connection to a remote object.

```
HttpURLConnection conn =(HttpURLConnection)url.openConnection();
```

Set properties for the HTTP POST request

Server-side push applications use a request method of `POST`.

```
conn.setRequestMethod("POST"); // post to BlackBerry Enterprise Server
```

To receive confirmation, set `doInput` to `true` to indicate that the application intends to read data from the URL connection.

```
conn.setDoInput(true);
```

To send data, set `doOutput` to `true` to indicate that the application intends to send data to the URL connection.

```
conn.setDoOutput(true);
```

Write data to the server connection

Invoke `getOutputStream()` to access an output stream. Write to the output stream, and then close it.

```
OutputStream out = conn.getOutputStream();
out.write(data);
out.close();
```

Read the server response

Invoke `getInputStream()` to access an input stream. Determine the size of the content and, if it is nonzero, open a data input stream, and then read in the content.

```
InputStream ins = conn.getInputStream();
int contentLength = conn.getContentLength();
if (contentLength > 0) {
   byte[] someArray = new byte [contentLength];
   DataInputStream dins = new DataInputStream(ins);
   dins.readFully(someArray);
   System.out.println(new String(someArray));
}
ins.close();
```

Disconnect the connection

Invoke `disconnect()` to indicate that the application plans to make no further requests to the server.

```
conn.disconnect();
```

Code example

The HTTPPush.java sample application, which is written using standard Java, sends a single .png image to a listening client application on the handheld. The application pushes data based on an email address. To test push applications with the simulator, define a mapping between the email address and the simulator PIN (2100000A).

The following code compiles using J2SE 1.4.2.

Example: HTTPPushServer.java

```
/*
 * HttpPushServer.java
 * Copyright (C) 2001-2004 Research In Motion Limited. All rights reserved.
 */

package com.rim.docs.samples.httppush;

import java.io.*;
import java.net.*;
import java.util.*;

public class HTTPPushServer {

    //constants
    private static final String HANDHELD_EMAIL = "scott.tooke@rim.com";
    private static final String HANDHELD_PORT = "6234";
    private static final String BES_HOST = "localhost";
    private static final int BES_PORT = 8080;
    private static final String CONTENT = "com/rim/docs/samples/httppush/logo.png";

    //constructor
    public HTTPPushServer() {
    }

    private static URL getPushURL(String HandheldEmail) {
        URL _pushURL = null;
        try {
            if ((HandheldEmail == null) || (HandheldEmail.length() == 0)) {
                HandheldEmail = HANDHELD_EMAIL;
            }
            _pushURL = new URL("http", BES_HOST, BES_PORT,
                "/push?DESTINATION="+ HandheldEmail
                +"&PORT="+HANDHELD_PORT+"&REQUESTURI=/");
        } catch (MalformedURLException e) {
            System.err.println(e.toString());
        }
        return _pushURL;
    }
    public static void postData(byte[] data) {
        try {
            URL url = getPushURL(HANDHELD_EMAIL);
            System.out.println("Sending to" + url.toString());
            HttpURLConnection conn = (HttpURLConnection)url.openConnection();
            conn.setDoInput(true); //for receiving the confirmation
            conn.setDoOutput(true); //for sending the data
            conn.setRequestMethod("POST"); //post the data to the BES
            OutputStream out = conn.getOutputStream();
            out.write(data); //write the data
            out.close();
            InputStream ins = conn.getInputStream();
            int contentLength = conn.getContentLength();
            System.out.println("Content length: "+ contentLength);
            if (contentLength > 0) {
                byte[] someArray = new byte [contentLength];
                DataInputStream dins = new DataInputStream(ins);
                dins.readFully(someArray);
                System.out.println(new String(someArray));
            }
            ins.close();
            conn.disconnect();
        } catch (IOException e) {
            System.err.println(e);
        }
    }
    public static void main (String args[]) {
        try {
            File f = new File(CONTENT);
            if ( f == null ) {
                throw new RuntimeException("Unable to Open File");
            }
            FileInputStream fi = new FileInputStream(f);
            if ( null == fi ) {
                throw new RuntimeException("Unable to open file");
            }
            int size = fi.available();
            byte[] imageData = new byte[size];
            int bytesRead = fi.read(imageData);
            fi.close();
            postData(imageData);
        } catch (IOException e) {
            System.err.println(e.toString());
        }
    }
}
```

Troubleshooting push applications

Push applications identify handhelds based on their email address. If users stop receiving data from a push application after they switch to a different handheld, it might indicate that the mapping between users' email addresses and handheld PINs is out-of-date. Verify that the BlackBerry Enterprise Server is operating correctly.

- BlackBerry Enterprise Server for Microsoft Exchange: The Mobile Data Service uses a Database Consistency tool, `dbconsistency.exe`, to maintain the mapping between email addresses and handheld PINs. Administrators can configure the frequency at which this tool runs, and can also run this tool manually. See the *BlackBerry Enterprise Server for Microsoft Exchange Handheld Management Guide* for more information.
- BlackBerry Enterprise Server for IBM Lotus Domino: An agent synchronizes the BlackBerry Directory with the BlackBerry Profiles database, in which the email-to-PIN mapping is stored. Administrators can configure the frequency at which the agent runs (by default, it runs every 5 minutes). When multiple BlackBerry Enterprise Servers are deployed, administrators must set up an appropriate replication schedule so that user information remains synchronized between each Mobile Data Service in the domain. At a minimum, you should replicate the BlackBerry Directory Database (`bbdir.nsf`). See the *BlackBerry Enterprise Server for IBM Lotus Domino Handheld Management Guide* for more information.

Packaging and deployment

Deploying applications using the BlackBerry Desktop Software

The Application Loader tool, which is part of the BlackBerry Desktop Software, uses an application loader (.alx) file to load new applications onto the handheld.

Create an application loader (.alx) file for each application, and then distribute the .alx and .cod files to users. See the *Application Loader Online Help* for more information.

Create application loader files

1. In the IDE, select a project.
2. On the **Project** menu, click **Generate .alx File**.

Distribute this .alx file with the .cod files for the application to users. When users connect their handheld to the computer, they can use the BlackBerry Desktop Software to load the application onto their handheld.

Note: By default, the .cod files for the application must exist in the same folder as the .alx file. If you change the location of .cod files relative to the .alx file, edit the .alx file and add the `<directory>` element to specify the file location. See " Appendix A: format of .alx files" on page 145 for more information.

Deploying applications wirelessly

The BlackBerry Handheld Software enables users to download applications wirelessly using the BlackBerry Browser. Users can download both standard MIDlets and BlackBerry applications. For users to download the application wirelessly, you must provide an application descriptor (.jad) with the appropriate parameters, and the .cod or .jar files for the application. In the BlackBerry Browser, the user selects the .jad file to download the application.

System administrators can set IT policies to control the use of third-party applications. See " Controlling application downloads" on page 119 for more information.

Make BlackBerry or MIDlet applications available for users to download wirelessly in one of the following ways:

- Use the Mobile Data Service to convert .jar files to .cod files.
- Use the BlackBerry JDE, which generates .cod files, to build your projects.

Deploy .jar files

The Mobile Data Service feature of the BlackBerry Enterprise Server provides a built-in transcoder to convert .jar files to .cod files, which enables users to download standard MIDlets. For example, corporate administrators can maintain a list of approved MIDlets on an Intranet site. Users can browse to the web page and select .jad files for applications to download. The BlackBerry Enterprise Server converts .jar files to .cod files before sending them to the handheld.

Note: The web server must set the MIME type for both .cod files and .jad files. For .cod files, the MIME type is `application/vnd.rim.cod`. For .jad files, the MIME type is `text/vnd.sun.j2me.app-descriptor`.

The following versions of the BlackBerry Enterprise Server provide the transcoder to convert .jar files to .cod files:

- BlackBerry Enterprise Server version 3.6 or later for Microsoft Exchange
- BlackBerry Enterprise Server version 2.2 or later for IBM Lotus Domino

Note: Users can only download .jar files if they access the network using the BlackBerry Enterprise Server with the Mobile Data Service enabled. The Mobile Data Service converts .jar files to the .cod file format that the handheld requires. If users access the network using a WAP gateway, they can only download .cod files.

MIDlet application properties

Application descriptor files have an extension of .jad. A standard MIDlet .jad file contains the following predefined attributes, and may contain additional applications defined by the application.

Required MIDlet attribute	Description
`MIDlet-Jar-Size`	The number of bytes in the .jar file.
`MIDlet-Jar-URL`	The URL from which the .jar file can be loaded.
`MIDlet-Name`	The name of the MIDlet suite.
`MIDlet-Vendor`	The organization that provides the MIDlet suite
`MIDlet-Version`	The version of the MIDlet suite, formatted as <major>.<minor>.<micro>.

Optional MIDlet attribute	Description
`MIDlet-Data-Size`	The minimum number of bytes of persistent data required by the MIDlet suite. The default is zero.
`MIDlet-Delete-Confirm`	A text message provided to the user when prompted to confirm deletion of the MIDlet suite.
`MIDlet-Description`	A description of the MIDlet suite.
`MIDlet-Icon`	The name of the .png image file within the .jar file used to represent the MIDlet suite.
`MIDlet-Info-URL`	A URL for further information describing the MIDlet suite.
`MIDlet-Install-Notify`	A URL to which a POST request is sent to confirm successful installation of the MIDlet suite.

Deploy .cod files

If you write BlackBerry applications using the BlackBerry JDE version 4.0, the BlackBerry JDE creates the required BlackBerry application descriptor (.jad) file when you build the project. You can also use the BlackBerry JDE to convert MIDlet .jar files to the .cod file format.

Make the .cod and .jad files available on a web server for users to download. By making .cod files available, you can deploy applications to users who do not access the network using a BlackBerry Enterprise Server.

Note: The web server must set the MIME type for both .cod files and .jad files. For .cod files, the MIME type is `application/vnd.rim.cod`. For .jad files, the MIME type is `text/vnd.sun.j2me.app-descriptor`.

BlackBerry application properties

In addition to the MIDlet application properties, the following attributes apply to BlackBerry .jad files.

Required RIM attribute	Description
`RIM-COD-Creation-Time`	The creation time of the .cod file.
`RIM-COD-Module-Dependencies`	A list of modules that the .cod file requires.
`RIM-COD-Module-Name`	The name of the module contained in the .cod file.
`RIM-COD-SHA1`	The SHA1 hash of the .cod file
`RIM-COD-Size`	The size (in bytes) of the .cod file.
`RIM-COD-URL`	The URL from which the .cod file can be loaded.

Optional RIM attribute	Description
`RIM-Library-Flags`	Reserved for use by RIM.
`RIM-MIDlet-Flags`	Reserved for use by RIM.
`RIM-MIDlet-NameResourceBundle`	The name of the resource bundle that the application depends on.
`RIM-MIDlet-Position`	The suggested position of the application icon on the Home screen. Note that this may not be the actual position of the application icon on the Home screen.

The BlackBerry JDE enables you to create a dual-purpose .jad file to support the downloading of MIDlets onto BlackBerry handhelds and other wireless devices. To do this, create a .jad file that contains both the `RIM-COD-URL` and `RIM-COD-Size` entries and `MIDlet-Jar-URL` and `MIDlet-Jar-Size`. On BlackBerry handhelds, the .cod files are downloaded; on other devices, the .jar files are downloaded.

Testing and debugging

Test applications

Test applications by running them in the handheld simulator or on a connected handheld. See " Using the debugging tools" on page 141 for more information.

1. In the IDE, on the **Debug** menu, click **Go**.
2. Use the application in the simulator or on a handheld.
3. In the IDE, on the **Debug** menu, click **Break Now**.

   Use the debugging tools on the **View** menu to retrieve detailed information. See " Using the debugging tools" on page 141 for more information.
4. To resume running the applications, on the **Debug** menu, click **Continue**.
5. To finish debugging, on the **Debug** menu, click **Stop Debugging**.

Testing applications using the simulator

In the IDE, the handheld simulator starts automatically when you run applications.

Action	Procedure
Roll the trackwheel	Roll the wheel button on your mouse, or press the UP ARROW and DOWN ARROW keys on your keyboard.
Click the trackwheel	Click the wheel button or press ENTER.
Run an application	Select the appropriate icon and click the wheel button or press ENTER.
Press keys	Press the keys on your keyboard.

Using the email service simulator

The BlackBerry JDE includes an email server simulator (ESS) that enables you to send and receive email between the handheld simulator and either a desktop email application, such as Microsoft® Outlook® Express, or POP3 and SMTP servers. A BlackBerry Enterprise Server is not required.

1. On the taskbar, click **Start** > **Programs** > **Research In Motion** > **BlackBerry Java Development Environment 4.0** > **Email Server Simulator**.
2. Select one of the following modes:
   - **Standalone mode**: The ESS stores messages on the local file system and communicates directly with a desktop email application. No POP3 or SMTP server is required.

     The ESS can communicate with any desktop email application that supports POP3 and SMTP, such as Outlook Express. The desktop email account must have the POP3 server set to localhost on port 110 and the SMTP server set to localhost on port 25.

- **Connected mode**: The ESS polls the user POP3 email server for incoming messages, and uses the user SMTP server to send messages. Valid POP3 and SMTP servers are required.

3. If you select the **Standalone mode** option, click **Clean FS** to erase ESS messages that are stored on the local file system.
4. If you select the **Connected mode** option, type the following information in the appropriate fields:
   - **Outgoing**: Host name of the SMTP server that your email account uses.
   - **Incoming**: Host name of the POP3 server that your email account uses.
   - **User name**: User name with which to connect to your email account.
   - **Password**: Password with which to connect to your email account.
   - **Poll inbox**: Specifies, in seconds, how often the simulator checks your email Inbox for new messages.
5. Type information in the following fields:
   - **Name**: Name to display in outgoing messages from the handheld simulator.
   - **Email**: Email address to display in outgoing messages from the handheld simulator.
   - **PIN**: Personal information number (PIN) that is used by the handheld simulator (default PIN is 0x2100000A).
6. Click **Launch**.

   If you change parameter values in the ESS window, a dialog box prompts you to save your changes.
7. Check the command prompt window for detailed information on ESS startup, including any login errors.

   After the ESS starts, use applications in the handheld simulator to send and receive email messages with a desktop email account.

Note: If you start the handheld simulator from a command prompt, specify the `/rport=0x4d4e` parameter to communicate with the email server simulator.

Testing an application that uses synchronization in the simulator

1. Exit the BlackBerry Desktop Software.
2. Connect a null modem cable between COM port 1 and COM port 2 on your computer.
3. In the IDE, on the **Edit** menu, click **Preferences**.
4. In the **Preferences** window, click the **Basic** tab.
5. Select the **Set serial port for device(s)** option, and then type **1**.
6. Click **OK**.
7. Build and run the application in the simulator.
8. After the simulator starts, start the BlackBerry Desktop Software.
9. In the BlackBerry Desktop Manager, on the **Options** menu, click **Connection Settings**.
10. Click **Detect** to detect the simulator.

    If the BlackBerry Desktop Software does not detect the simulator, restart your computer and repeat steps 7 through 10.

Testing applications using a connected handheld

When a handheld is connected to a computer, run applications on the handheld and use the IDE debugging tools to perform testing and optimization.

Note: To attach the IDE to a handheld that is connected to a serial port, install the Java Communications API, version 2.0. Download the API from http://java.sun.com/products/javacomm/. This API is not required if the handheld is connected to a USB port.

Install .debug files

To debug applications using a handheld, the .debug files in the BlackBerry JDE must match the software build version of the handheld.

1. Download the .debug files for your handheld build version from the BlackBerry Developer Zone.
2. In the IDE, on the **Edit** menu, click **Preferences**.
3. Click the **Debug** tab.
4. Click the **Other** tab.
5. In the **Handheld debug file location** field, type the location of the downloaded .debug files.

Load an application for testing

Warning: If your handheld contains any important information, such as messages or contacts, back up this data prior to loading an application for testing.

The JavaLoader.exe tool enables you to add or update applications on a handheld using a command prompt. Use this tool for development and testing purposes only. For production applications, use the BlackBerry Desktop Software.

Note: You must load applications with dependencies in the correct order. If project A is dependent on project B, load the project B .cod file before loading project A.

1. Exit the BlackBerry Desktop Software.
2. Connect the handheld to the computer.
3. At the command prompt, switch to the bin folder in the JDE installation folder and run the following command:

   ```
   JavaLoader [-usb] [-pport] [-bbps] [-wpassword] load files
   ```

Option	Description
port	COM port to which the handheld is connected (default is 1), or the handheld PIN if the handheld is connected to a USB port (the **-usb** option must also be specified)
bps	bit rate speed to the serial port (default is 115200)
password	specifies the password for your handheld, if you have set one
files	one or more .cod file names, separated by a space, to load onto the handheld

Delete applications from the handheld

At the command prompt, run the following command, where the -f option forces removal of the application even if it is in use:

```
JavaLoader [-usb] [-pport] [-bbps] [-wpassword] erase [-f] files
```

Attach the IDE debugger

1. To attach the IDE debugger to a handheld that is connected to a USB port, start BBDevMgr.exe. The BBDevMgr.exe tool is installed with the BlackBerry Desktop Software version 3.5.1 or later in C:\Program Files\Common Files\Research In Motion\USB Drivers.

2. Perform one of the following actions:
   - For a handheld that is connected to a serial port, click **Attach to** > **Handheld** > **COM n**, where n is the serial port to which your handheld is connected.
   - For a handheld that is connected to a USB port, click **Attach to** > **Handheld** > **USB (PIN)**, where PIN is the PIN of a connected handheld.

Testing HTTP network connections

To test applications that require an HTTP network connection, use the Mobile Data Service simulator, which is included in the BlackBerry JDE.

On the taskbar, select **Start** > **Programs** > **Research In Motion** > **BlackBerry Java Development Environment Version 4.0** > **MDS Simulator**.

Tip: To configure the IDE to start the Mobile Data Service when the simulator launches, in the IDE, on the **Edit** menu, click **Preferences**. Click the **Simulator** tab. Select **Launch Mobile Data Service (MDS) with simulator**.

Use a WAP gateway

You can set up an HTTP connection using a WAP gateway that is hosted by your service provider. BlackBerry handhelds support WAP 1.1 features.

Note: WAP service is not supported on all wireless networks. Before you start development, contact the service provider for information on how to connect to their WAP gateway.

To set up an HTTP connection using WAP, include the `WAPGatewayIP` and `WapGatewayAPN` parameters in `Connector.open()` at the end of the URL.

```
Connector.open("http://host;WAPGatewayIP=127.0.0.1; WAPGatewayAPN=rim.net.gprs");
```

WAP parameters are separated by a semicolon (;). They must not include spaces.

Parameter	Description	Default
`WapGatewayIP`	IP address of the gateway	–
`WapGatewayAPN`	access point name (for GPRS networks only); for testing purposes, use rim.net.gprs	–
`WapGatewayPort`	gateway port value (if port 9203 is specified, WTLS is used unless `WapEnableWTLS`=false is specified)	9201
`WAP_GATEWAY_PORT_DEFAULT`		
`WapSourceIP`	IP address of the source	127.0.0.1
`WAP_SOURCE_IP_DEFAULT`		
`WapSourcePort`	source port value	8205
`WAP_SOURCE_PORT_DEFAULT`		
`TunnelAuthUsername`	user name for APN session, when PAP or CHAP authentication is used	none
`TunnelAuthPassword`	password for APN session, when PAP or CHAP authentication is used	none
`WapEnableWTLS`	enables or disables WTLS (if this parameter is not specified, WTLS is used by default for connections to port 9203); the handheld supports WTLS Class 1 (encryption only, no authentication) and Class 2 (encryption and server authentication)	none

Note: In the handheld simulator, when you test applications that require a WAP connection, add the command line option `/rport=<wap_source_port>`, typically `/rport=8205`. The APN of the simulator is rim.net.gprs.

In the IDE, on the **Edit** menu, click **Preferences**. Click the **Simulator** tab, and then click the **Advanced** tab. Add the necessary command line option to the Simulator Command Line field.

Configure the Mobile Data Service simulator

1. Open the rimpublic.property file (located in the MDS\config folder) in a text editor.
2. Edit parameters to configure the following features:
   - See " Logging level parameters" on page 140 for more information.
   - See " HTTP support parameters" on page 140 for more information.
   - See " HTTPS support parameters" on page 141 for more information.
   - See " Push support parameters" on page 141 for more information.
   - See " Email-to-PIN mapping" on page 141 for more information.
3. Restart the Mobile Data Service simulator for changes to take effect.

Note: In a production environment, the BlackBerry Enterprise Server system administrator configures the Mobile Data Service parameters using the BlackBerry Manager. Contact your system administrator for more information.

Logging level parameters

Parameter	Description	Default
`Logging.level`	This parameter specifies the type of information that is written to the logs, if logging is enabled. • 1: writes only information on events, such as Mobile Data Service start or stop • `2:` writes events and errors • 3: writes events, errors, and warnings • 4: writes events, errors, warnings, and debugging information	4
`Logging.console.log.level`	This parameter specifies the type of information that appears in the console, if logging is enabled. See the description for the `Logging.level` parameter.	4

HTTP support parameters

Parameter	Description	Default
`application.handler.http.logging`	This parameter enables (TRUE) or disables (FALSE) HTTP standard logging (HTTP headers only).	TRUE
`application.handler.http.logging.verbose`	This parameter enables (TRUE) or disables (FALSE) HTTP debug logging (HTTP data and headers). This parameter should be set to TRUE only when necessary to debug a specific problem.	FALSE
`application.handler.http.CookieSupport`	This parameter enables (TRUE) or disables (FALSE) cookie storage. If you select TRUE, the Mobile Data Service manages cookie storage instead of the handheld. This reduces the load on the handheld significantly.	TRUE
`application.handler.http.AuthenticationSupport`	This parameter enables (TRUE) or disables (FALSE) storage of user authentication information.	TRUE
`application.handler.http.AuthenticationTimeout`	If HTTP authentication is set to TRUE, this parameter determines the length of time (in milliseconds) before the authentication information becomes invalid. This timer resets whenever the user issues a request that invokes the authentication information for a particular domain.	3600000
`application.handler.http.device.connection.timeout`	This parameter sets the length of time (in milliseconds) before a connection attempt to the handheld expires if the handheld is unreachable.	140000
`application.handler.http.server.connection.timeout`	This parameter sets the length of time (in milliseconds) before a connection attempt to a server expires if the server is unreachable.	150000

HTTPS support parameters

Parameter	Description	Default
`application.handler.https.logging`	This parameter enables (TRUE) or disables (FALSE) HTTPS logging for testing purposes.	TRUE
`application.handler.https.allowUntrustedServer`	This parameter allows the Mobile Data Service to connect to untrusted servers (TRUE), or restricts access to trusted servers only (FALSE). A server is trusted if its certificate is installed on the Mobile Data Service host machine. See " Install certificates using the keytool" on page 151 for more information.	FALSE

Push support parameters

Do not change these parameters.

Parameter	Description	Default
WebServer.listen.host	This parameter defines the computer on which the Mobile Data Service listens for HTTP POST requests from push applications.	`localhost`
`WebServer.listen.port`	This parameter defines the port on which the Mobile Data Service listens for HTTP POST requests from push applications.	8080

Email-to-PIN mapping

In a production environment, the BlackBerry Enterprise Server automatically maps user email addresses to the personal identification number (PIN) of their handhelds. In the BlackBerry JDE, you can simulate the mapping between email addresses and PINs.

Tip: You only need to configure email to PIN mappings if you are testing a push application. See " Creating client/server push applications" on page 121 for more information.

In the `rimpublic.property` file, add or change entries in the `[Simulator]` section. Entries use the following format:

```
Simulator.<PIN>=<host>:<port>, <email_address>
```

For example:

```
Simulator.2100000a=localhost:81, user2100000a@pushme.com
```

Change the email address so that you can test push applications that use actual email addresses. Pushed data is sent to the specified handheld simulator.

The default PIN for the simulator is 2100000a. To change the simulator PIN, set the `/rsim` option. In the IDE, on the **Edit** menu, click **Preferences**. Click the **Simulator** tab, and then click the **Advanced** tab. In the **Simulator Command Line** field, change **`/rsim=0x2100000A`**.

The port must match the value set in the `IPPP.push.listen.tcp.port` parameter. The default is 81.

Using the debugging tools

Note: This section provides an overview of some of the debugging tools that are available in the IDE. See the *IDE Online Help* for detailed information on using the IDE.

Analyze code coverage

To display a summary of code that has been run, use the coverage tools. A summary is useful when you design and run test cases because you can see exactly what has been tested.

1. Set two or more breakpoints in your code.
2. In the IDE, on the **View** menu, click **Coverage**.
3. To reset information to 0, in the coverage pane, click **Clear**.
4. Run your application to the next breakpoint.
5. To display the percentage of code that has been run since you clicked **Clear**, in the coverage pane, click **Refresh**.

Using the profiler

Use the IDE profiler tool to optimize your code. The profiler tool displays the percentage of time that is spent in each code area, up to the current point of execution.

Note: To improve the reliability of results when you run the profiler, exit other Microsoft Windows applications.

Run the profiler

1. At the start of a section of code, press **F9** to set a breakpoint.
2. At the end of a section of code, press **F9** to set a breakpoint.
3. On the **Debug** menu, click **Go**.
4. Use the application in the simulator to run the appropriate code until it reaches the breakpoint.
5. On the **View** menu, click **Profile**.
6. In the profile pane, click **Options**.
7. Select the type of method attribution, a sorting method, and the type of information to profile. See " Set profile options" on page 143 for more information.
8. Click **OK**.
9. To remove the profiler data and reset the running time, in the profile pane, click **Clear**.
10. On the **Debug** menu, click **Go**.
11. Use the application in the simulator to run the appropriate code until it reaches the breakpoint.
12. If the profile pane is not visible, on the **View** menu, click **Profile**.
13. To retrieve all accumulated profile data, on the profile pane, click **Refresh**.
14. Click **Save** to save the contents of the profile pane to a comma separated values (.csv) file.

Profile View	Description
Summary	The Summary view displays general statistics about the system and the garbage collector. It displays the percentage of time that the Java VM has spent idle, running code, and performing quick and full garbage collection. The Percent column displays the percentage of total VM running time, including idle and collection time.
Methods	The Methods view displays a list of modules, sorted either by the information that you are profiling or by the number of times that each item was run.
Source	The Source view displays the source lines of a single method. This enables you to navigate through the methods that call, and are called by, that method. Click the **Back** and **Forward** buttons to follow the history of methods that you have visited in the Source view. In this view, the Percent column displays the percentage of total VM running time, not including idle and garbage collection time.

Set profile options

1. Click the **Options** tab.
2. From the **Method attribution** drop-down list, select one of the following options:
   - To calculate the amount of time that is spent running bytecode in a method and methods that are invoked by that method, select **Cumulative.**
   - To calculate the amount of time spent executing bytecode in that method only, select **In method only**. The timer stops when a call is made to another method.
3. From the **Sort method by** drop-down list, select **Count** to sort methods in the Profile views by the number of times that the item was run, or select the other option to sort methods according to the data that is being profiled.
4. From the **What to profile** drop-down list, select the type of data to profile.

Finding memory leaks

Use the memory statistics and objects IDE tools to find and correct memory leaks.

Use the Memory Statistics tool

The Memory Statistics tool displays statistics on the number of objects and bytes that are used for object handles, RAM, and flash memory.

1. On the **View** menu, click **Memory statistics**.
2. Set two or more breakpoints in your code.
3. Run your application to the first breakpoint.
4. Click **Refresh** to refresh the memory statistics.
5. Click **Snapshot** to take a snapshot.
6. Run the application to the next breakpoint.
7. Click **Refresh**.
8. Click **Compare to Snapshot**.
9. To save the contents of the memory statistics pane to a comma separated values (.csv) file, click **Save**.

Use the Objects tool

The Objects tool displays all of the objects in memory to help you locate object leaks. Object leaks can cause the VM to run out of flash memory, which resets the handheld.

1. In the IDE, on the **Debug** menu, click **Go**.
2. On the **Debug** menu, click **Break Now**.
3. On the **View** menu, click **Objects**.
4. Click **GC**.
5. Click **Snapshot**.
6. On the **Debug** menu, click **Continue**.
7. Complete tasks in the application that should not increase the number of reachable objects; for example, create a new contact, and then delete it.
8. On the **Debug** menu, click **Break Now**.

9. In the objects pane, click **GC**.

   Click **Compare to Snapshot**. The objects pane displays the number of objects that were deleted and added since the previous snapshot. If the number of objects that are added is not the same as the number of objects that are deleted, you might have an object leak. Use the **Type** and **Process** filters to view specific objects.
10. To save the contents of the objects pane to a comma separated values (.csv) file, click **Save**.

Appendix A: format of .alx files

.alx files

The Application Loader tool, which is part of the BlackBerry Desktop Software, uses an application loader (.alx) file to load new applications onto the handheld. Use the IDE to generate .alx files for your projects.

The following information is provided only as a supplementary reference. In most cases, you do not need to edit the .alx files generated in the IDE.

In a text editor, you can edit .alx files that the IDE generates. The .alx file uses an XML format.

Example: Sample .alx file

```
<loader version="1.0">
   <application id="com.rim.samples.device.httpdemo">
   <name>Sample Network Application</name>
   <description>Retrieves a sample page over HTTP connection.
   </description>
   <version>1.0</version>
   <vendor>Research In Motion</vendor>
   <copyright>Copyright 1998-2003 Research In Motion</copyright>
   <language langid="0x000c">
      <name>Application D'Échantillon</name>
      <description>Obtenir une page du réseau
      </description>
   </language>
   <fileset Java="1.0">
      <directory>samples/httpdemo</directory>
      <files>
         net_rim_httpdemo.cod
         net_rim_resource.cod
         net_rim_resource__en.cod
         net_rim_resource__fr.cod
      </files>
   </fileset>
   </application>
</loader>
```

Nesting modules

Create a nested structure in an .alx file to provide optional components for an application. Typically, nested modules provide optional features that are not applicable to all users. Users can choose whether to install optional modules.

Nesting creates an implicit dependency for nested modules on the base application. To define an explicit dependency on another application or library, use the `<requires>` tag. See " Elements in .alx files" on page 148 for more information.

Example: Sample .alx file for an application with a nested module

```
<loader version="1.0">
   <application id="net.rim.sample.contacts">
      <name>Sample Contacts Application</name>
   <description>Provides the ability to store a list of contacts.
      </description>
      <version>1.0</version>
      <vendor>Research In Motion</vendor>
      <copyright>Copyright 1998-2001 Research In Motion</copyright>
      <fileset Java="1.0">
         <directory>samples/contacts</directory>
         <files>
            net_rim_contacts.cod
            net_rim_resource.cod
            net_rim_resource__en.cod
            net_rim_resource__fr.cod
         </files>
      </fileset>
      <application id="net.rim.sample.contacts.mail">
         <name>Sample Module for Contacts E-Mail Integration</name>
         <description>Provides the ability to access the email
            applicaton</description>
         <version>1.0</version>
         <vendor>Research In Motion</vendor>
         <copyright>Copyright 1998-2001 Research In Motion</copyright>
         <fileset Java="1.0">
            <directory>samples/contacts</directory>
            <files>
               net_rim_contacts_mail.cod
            </files>
         </fileset>
      </application>
   </application>
</loader>
```

Specifying a handheld version

Applications that use APIs only available on particular versions of the handheld software should specify supported handheld versions using the `_blackberryVersion` attribute.

Specify a range using the following rules:

- Square brackets [] indicate inclusive (closed) range matching.

- Round brackets () indicate exclusive (open) matching.
- Missing lower ranges imply 0.
- Missing upper ranges imply infinity.

For example, (4.0,] indicates any version between 4.0 and infinity.

The following example prevents modules from loading on versions of the handheld software earlier than 4.0.

```
<application id="<application_id>" _blackberryVersion="[4.0,)">
...
</application>
```

The following example provides alternate modules for different versions of the handheld software.

```
<application id="<application_id>">
   ...
   <fileset _blackBerryVersion="(,4.0)">
   ... modules for handheld software versions earlier than 4.0
   </fileset>
   <fileset _blackBerryVersion="[4.0,)">
   ... modules for handheld software versions 4.0 and later
   </fileset>
</application>
```

Elements in .alx files

Element	Attributes	Description
`loader`	`version`	The `loader` element contains one or more `application` elements. The `version` attribute specifies the version of the Application Loader.
`application`	`id`	The `application` element contains the elements for a single application. The `application` element can also contain additional nested `application` elements. Nesting enables you to require that when an application is loaded, its prerequisite modules are also loaded. The `id` attribute specifies a unique identifier for the application. To provide uniqueness, use an ID that includes your company domain, in reverse. For example, `com.rim.samples.docs.helloworld`.
`library`	`id`	The `library` element can be used instead of the `application` tab. It contains the elements for a single library module. No nested modules are permitted. By default, a library module is hidden so that it does not appear in the Application Loader. Typically, use the library element as the target of a <requires> element, so that when a particular application is loaded onto the handheld, a required library is also loaded. **Note**: This element is supported in BlackBerry Desktop Software version 3.6 or later.
`name`	–	The `name` element provides a descriptive name for the application, which appears in the Application Loader. It does not appear on the handheld.
`description`	–	The `description` element provides a brief description of the application, which appears in the Application Loader. It does not appear on the handheld.
`version`	–	The `version` element provides the version number of the application, which appears in the Application Loader. This version number is for information purposes only.
`vendor`	–	The `vendor` element provides the name of the company that created the application, which appears in the Application Loader.
`copyright`	–	The `copyright` element provides copyright information, which appears in the Application Loader.
`required`	–	The `required` element enables you to force an application to be loaded. In the Application Loader, the application is selected for installation, and the user cannot change this. Add the following line: `<required>true</required>`. The `required` tag should be used by corporate system administrators only; it is not intended for use by third-party software vendors. **Note**: This element is supported in BlackBerry Desktop Software version 3.5 or later.
`hidden`	–	The `hidden` element hides a package so that it does not appear to users in the Application Loader. Add the following line: `<hidden>true</hidden>`. Use this element in conjunction with the `required` element to load the application by default, or set the `requires` tag to load this package if another application is loaded. The `hidden` tag should be used by corporate system administrators only; it is not intended for use by third-party software vendors. **Note**: This element is supported in BlackBerry Desktop Software version 3.6 or later.
`language`	`langid`	The `language` tag enables you to override the text that appears in the Application Loader when the Application Loader is running in the language specified by the `langid` attribute. To support multiple languages, specify multiple `language` tags. To specify the `name`, `description`, `version`, `vendor`, and `copyright` tags for each language, nest these tags in the `language` tag. If you do not nest a tag, the text appears in the default language. The `langid` attribute specifies the Win32 `langid` code for the language to which this information applies. For example, some Win32 `langid` codes are: `0x0009` (English), `0x0007` (German), `0x000a` (Spanish), `0x000c` (French).

Element	Attributes	Description
`requires`	`id`	The `requires` element is an optional element that specifies the `id` of a package on which this application depends. This element can appear more than once, if the application depends on more than one other application. When an application is loaded onto the user handheld, all packages that are specified by the `<requires>` tag are also loaded. **Note**: This element is supported in BlackBerry Desktop Software version 3.6 or later.

`fileset`	`Java`
	`radio`
	`langid`
	`color`
		The `fileset` element includes an optional `directory` element and one or more `files` elements. It specifies a set of .cod files, in a single directory, to load onto the handheld. To load files from more than one directory, include one or more `fileset` elements in the .alx file.
		The `Java` attribute specifies the minimum version of the BlackBerry Java VM with which the .cod files are compatible. The current VM is version 1.0. The `Java` attribute is required.
		The `radio` attribute enables you to load different applications or modules depending on the network type of the handheld. Possible values include `Mobitex`, `DataTAC`, `GPRS`, `CDMA`, and `IDEN`. The `radio` attribute is optional.
		The `langid` attribute enables you to load different applications or modules depending on the language support that users add to the handheld. The value is a Win32 `langid` code; for example: `0x0009` (English), `0x0007` (German), `0x000a` (Spanish), `0x000c` (French). The `langid` attribute is optional.
		The `color` attribute enables you to load different applications or modules for color or monochrome displays. The value is a `Boolean`; `true` means color display and `false` means monochrome.

`directory`	–	The `directory` element provides the location of a set of files. The `directory` element is optional. If you do not specify a `directory`, the files must be in the same location as the .alx file. The directory is specified relative to the location of the .alx file.
`files`	–	The `files` element provides a list of one or more .cod files, in a single directory, to load onto the handheld for an application.

Appendix B: Mobile Data Service reference

HTTP requests

A client opens a connection and sends an HTTP request message to a server. The server sends a response message, which usually contains the requested resource.

```
<method> <resource_path><version>
Header1: value1
Header2: value2
Header3: value3
<optional message>
```

HTTP request variable	Definition
`method`	Method names indicate an action, such as GET, HEAD, or POST. The commonly used method is GET, which requests a resource from a server.
`resource_path`	The path that points to the requested resource is the part of the URL that appears after the host name. This is also called the *Request URL*.
`version`	The version of HTTP that you are running, and is noted as "HTTP/x.x." The BlackBerry Enterprise Server supports versions 1.0 and 1.1.
`header`	The header provides information about the request or about any object that is sent in the message body.
`optional message`	The HTTP message can contain data. In a request, this is where user-entered data or uploaded files are sent to the server. When an object accompanies the message, the request usually also includes headers that define its properties.

HTTP responses

Upon receipt of an HTTP request message, the server sends a response message, which usually contains the requested resource.

```
<HTTP version><status_code><reason>
Header1: value1
Header2: value2
Header3: value3
```

`<message>`

HTTP response variable	Definition
`HTTP_version`	This variable indicates the version of HTTP that you are running is noted as "HTTP/x.x." The BlackBerry Enterprise Server supports versions 1.0 and 1.1.
`status_code`	The status code is a numerical value that reflects the results of the initial request of the client. For example, `200 (OK)` indicates successful transmission; `404 (Not Found)` indicates that the requested URL could not be found.
`reason`	The `reason` is a text message that is associated with the status code.
`header`	The `header` provides information about the response and about the object that is being sent in the message body.
`message`	The HTTP message must contain data. In a response, this message provides content that was requested by the client; the response also includes headers that define its properties.

Tip: Applications should check the status code in HTTP response messages. Any code other than `200 (OK)` indicates that an error might have occurred when establishing the HTTP connection.

HTTPS support

To provide additional authentication and security if your application accesses servers on the Internet, set up a secure HTTP (HTTPS) connection over TLS.

Certificate management for HTTPS

When the handheld requests an HTTPS connection in proxy mode, the Mobile Data Service sets up the SSL connection on behalf of the handheld. System administrators configure the Mobile Data Service either to allow connections to untrusted servers, or to restrict access to trusted servers only. This configuration applies to connections in proxy mode only; in end-to-end mode, the handheld sets up the SSL connection.

In the BlackBerry Manager, administrators edit the Mobile Data Service properties and set TLS and HTTPS options. See the *BlackBerry Enterprise Server Administration Guide* for more information.

Tip: In the Mobile Data Service simulator, allow or deny access to untrusted servers by editing the rimpublic.property file. Set the `application.handler.https.allowUntrustedServer` parameter to true or false. See " Configure the Mobile Data Service simulator" on page 140 for more information.

A server is trusted if its certificate is installed in the Mobile Data Service.

Install certificates using the keytool

1. Save the certificate from the web site to a file.
2. Copy the certificate file to the jre1.4.1\lib\security folder on the computer on which the Mobile Data Service resides.
3. To import the certificate into the key store, use the keytool, which is installed in the JRE bin folder, such as C:\Program Files\Java\j2re1.4.1\bin. For example, type:

   ```
   keytool -import -file <cert_filename> -keystore cacerts
   ```

Visit http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html for more information on using the keytool.

Transcoders

The Mobile Data Service supports plug-in Java applications, called transcoders, that perform processing on data that is sent to and from the handheld.

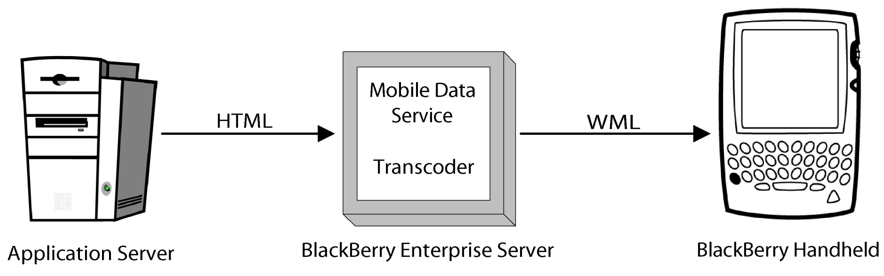
The Mobile Data Service provides the following transcoders by default:

- WML -> WMLC: Converts WML into a compact format
- WMLScript -> WMLScriptC: Converts WML Script into compact format
- JAD -> COD: Converts MIDlet applications into the BlackBerry format
- XML -> WBXML: Converts text content in eXtensible Markup Language (.xml) into the following WAP Binary XML (.wbxml) MIME type: application/vnd.wap.wbxml.
- SVG -> PME: Converts Scalable Vector Graphics Format(.svg) into Plazmic Media Engine binary file format (.pme).
- Image transcoder: Converts each of the following image types into the Portable Network Graphics format (.png):
  - Joint Photographic Expert Group (.jpeg)
  - Graphics Interchange Format (.gif)
  - Tagged Image File Format (.tiff)
  - Portable Graymap Format (.pgm)
  - Portable Pixmap Format (.ppm)
  - Portable Any Map Format (.pnm): this includes ASCII and binary .pbm, .pgm and .ppm files
  - Icon Format (.ico)
  - Wireless Bitmap Format (.wbmp)
  - Portable Bitmap Format (.pbm)
  - Wireless Bitmap Format (.wbmp)
  - Bitmap Format (.bmp)

Note: The image transcoder converts the above-mentioned image formats into the following PNG MIME type: image/vnd.rim.png.

You can also write your own transcoders to perform custom data processing. For example, you might write a transcoder if your handheld application is exchanging data with a server that is not designed for a mobile device, and the data that is sent from the server is in an inappropriate format for the handheld.

A transcoder can change the data format or remove extraneous information to reduce network traffic and to support a simplified application on the handheld. For example, you might write a transcoder to convert HTML content to WML.



Data transcoding process

If you write a server-side application specifically to support a custom application on the handheld, a transcoder is not required. You can design the server application to output data in a suitable format.

You can also perform data processing as part of a separate server-side process, before data is sent to the BlackBerry Enterprise Server.

Transcoder API

Main transcoder class	Description
`HttpContentTranscoder`	This class provides methods to control HTTP request and response contents and properties. All transcoders must extend this class and implement its abstract methods.
`HttpContentTranscoderException`	This exception is thrown to indicate that the transcoding process is not successful. The exception is forwarded to the handheld as an `IOException`.
`HttpHeader`	This class provides methods for manipulating header fields in HTTP requests and responses.
`HttpParameter`	This class represents an HTTP query parameter. It provides methods for retrieving and setting parameter names and values.
`HttpRequest`	This class extends the `HttpTransmission` class, and represents an HTTP request that contains headers, parameters, and content.
`HttpResponse`	This class extends the `HttpTransmission` class, and represents an HTTP response that contains headers, parameters, and content.
`HttpTransmission`	This class provides methods to retrieve and set content, headers, and parameters of HTTP requests and responses.

See the *API Reference* for more information.

Create an HTTP content transcoder

To create an HTTP content transcoder with full control over HTTP request and response content, extend the `HttpContentTranscoder` class. This transcoder can also modify, add, or remove the HTTP request and response properties.

The transcoder must be defined in the `net.rim.protocol.http.content.transcoder.<`*`name`*`>` package, where <*name*> is the identifier for the transcoder that is used in the mapping file. Its class name must be `Transcoder`.

The transcoder location is either specified in the `httpcontenttranscoders.property` file or by a handheld application.

Handling HTTP content transcoder exceptions

Any exception that a transcoder throws is sent to the handheld application as an `IOException`. The associated detailed message is also copied to the `IOException`.

Note: Instead of throwing an exception to the HTTP Connection Handler, a transcoder might also send an HTTP response to a handheld application, indicating that an internal server error has occurred.

Selecting transcoders

To determine whether a transcoder is required when the handheld requests content, the Mobile Data Service compares the type of content that is requested by the client with the type of content that the server responds with. If the content types are different, the Mobile Data Service checks the httpcontenttranscoder.property file to determine whether a transcoder is available to convert the content.

The x-rim-transcode-content header, in the HTTP request sent by the handheld, specifies the input format of the content that is to be converted by the Mobile Data Service. The content is converted if the following conditions are satisfied:

- The input format specified by the x-rim-transcode-content header is also specified in the Accept header of the HTTP request that the Mobile Data Service sends to the web server
- The web server responds with a content type, or output type that is different from the content type accepted by the handheld.
- The httpcontenttranscoder.property file contains a transcoder that can convert the content the web server sends back to the Mobile Data Service in a format that is accepted by the handheld.

To determine whether a transcoder is required when content is pushed, the Mobile Data Service checks the content-type and x-rim-transcode-content headers in the server-side push application. If the content-type and x-rim-transcode-content headers are the same, the Mobile Data Service converts the content specified in the headers to a format the handheld can display.

The httpcontenttranscoder.property file maps the input format to a given transcoder. The httpcontenttranscoder.property file specifies that the input format (.wml) in the example below, is converted to either .wbxml or .wmlc.

```
content-type: text/vnd.wap.wml
x-rim-transcode-content: text/vnd.wap.wml
```

You can also create custom transcoders that reformat or convert data that is requested by a handheld application. For example, you could write a transcoder to convert HTML content to WML. See " Creating transcoders" on page 157 for more information.

HTTP requests

The Mobile Data Service uses MIME types to determine whether the format of the content that is attached to a server HTTP response matches the format of the content that is requested by the handheld.

The Mobile Data Service examines the list of available transcoders and extends the `Accept` header. For example, if a request accepts a MIME type of WML and an HTML-to-WML transcoder exists, `HTML` is added to the `Accept` header. See " Mapping transcoders" on page 156 for more information.

In the HTTP request, the handheld application can include two headers to control the type of content that it receives:

- `Accept` header
- `Content-Transcoder` header

Header	Description

Accept header	In the `Accept` header of an HTTP request, the handheld application lists the MIME types that it accepts from the content server. For example, if the handheld application can accept content in WML or XML format, it sends the following header in its HTTP request:
	`Accept: text/wml, text/xml`
	By default, when the `Accept` header lists more than one MIME type, the Mobile Data Service assigns preferences from left to right, where the first MIME type listed is assumed to be preferred.
	The handheld application can also assign quality factors to MIME types to indicate preferences. Quality factors range from 0 (lowest preference) to 1 (highest preference). For example, the following header indicates that compiled WML (WMLC) is preferred but WML is also accepted.
	`Accept: text/vnd.wap.wml;q=0.5, application/vnd.wap.wmlc;q=1`

Content-Transcoder header	The Content-Transcoder header is a BlackBerry header that enables the application to specify a particular transcoder to apply to any content before it is returned to the handheld. Using the Content-Transcoder header overrides the default process that the Mobile Data Service uses to select a transcoder.

HTTP responses

The content server produces HTTP responses that include a `Content-Type` header. The `Content-Type` header indicates the MIME type for a particular packet of data. For example, when the content server returns content in HTML format, the HTTP response includes a `Content-Type` header with the value `text/html`.

The Mobile Data Service compares the request that is made by the handheld application to the response that is provided by the content server. The following sections demonstrate how Mobile Data Service determines whether a transcoder is required.

Response type	Description

No transcoding required	The handheld application sends a request for content that is formatted in WML. The request contains the following header:
	`Accept: text/vnd.wap.wml`
	In its response, the content server sends the following header:
	`Content-Type: text/vnd.wap.wml`
	Because the type of content that is returned by the server matches the type that is requested by the handheld, the Mobile Data Services does not need to perform any conversion. It forwards the content to the handheld unchanged.

Mobile Data Service performs transcoding	The handheld application sends a request for content. The request contains the following header:
	`Accept: text/vnd.wap.wml; q=0.5, application/vnd.wap.wmlc; q=1`
	This header indicates that the handheld application accepts content in either WML or compiled WML (WMLC) formats, but that WMLC is preferred.
	In its response, the content server sends the following header:
	`Content-Type: text/vnd.wap.wml`
	Because the type of content that is returned by the server does not match the preferred content type that the handheld requested, the Mobile Data Service searches for a transcoder that can convert the available MIME type (WML) into the preferred MIME type (WMLC).
	If an appropriate transcoder is not available, the Mobile Data Service forwards the WML content to the handheld unchanged, because the initial request indicated that the handheld application does accepts WML content.

Extending Accept headers

Each transcoder implements a method to create a hash table that maps the formats that the transcoder accepts as input from a content server and the formats that the transcoder can create as output.

The Mobile Data Service collects this information when it starts so that it can modify the `Accept` header field before it forwards an HTTP request from the handheld. For example, a handheld application sends an HTTP request with the following `Accept` header:

```
Accept: application/vnd.wap.wmlc, text/vnd.wap.wmlscriptc
```

After reviewing the transcoder mapping table, Mobile Data Service appends the following items to the `Accept` header line:

`application/vnd.wap.wmlscript`, `text/wml`, and `text/vnd.wap.wml`

```
Accept: application/vnd.wap.wmlc, text/vnd.wap.wmlscriptc, application/
   vnd.wap.wmlscript, text/wml, text/vnd.wap.wml
```

This extended `Accept` header now lists all MIME types that the content server can provide, if the transcoders are available to perform the required conversion before sending the content to the handheld.

Mapping transcoders

The httpcontenttranscoderslist.property file, located in the mdshome\config folder, specifies how the Mobile Data Service should manage the exchange of various content types between handheld applications and content servers.

The following is an example of the httpcontenttranscoderslist.property file:

```
text/vnd.wap.wml->application/vnd.wap.wmlc:vnd.wap.wml
text/vnd.wap.wml->application/vnd.wap.wbxml:vnd.wap.wml
text/vnd.wap.wmlscript->application/vnd.wap.wmlscriptc:wmls
text/html->application/vnd.rim.html.filtered:html
text/vnd.wap.wml:vnd.wap.wml
text/vnd.sun.j2me.app-descriptor->application/vnd.rim.cod:vnd.rim.cod
default:pass
```

Each entry uses the following format:

`InputType` `[->` *`OutputType`*`]:` *`Action`*

where:

- `InputType` is the MIME type that is available from the content server
- `OutputType` is the MIME type that the handheld requests
- `Action` is one of the following values:
  - transcoder package name, such as `vnd.wap.wml`
  - `Pass`: send data without change
  - `Discard`: discard data without sending it

The following sections explain the possible formats for entries in the httpcontenttranscoderslist.property file.

Format 1

An entry with the following format specifies the action that the Mobile Data Service perform when the type of content that is available from the server is different from the type of content that the handheld has requested:

```
InputType -> OutputType:Transcoder OR RSV
```

For example, the handheld application requests `text/wml`, but the content server only has `text/xml`. The Mobile Data Service finds this entry for this MIME type:

```
Text/xml -> Text/wml : vnd.wap.wml
```

According to this entry, the Mobile Data Service must use the `vnd.wap.wml` transcoder to convert XML content to WML.

Format 2

An entry with the following format specifies the action that the Mobile Data Service performs when it receives content of a given type from the server, regardless of what the handheld has requested:

```
InputType:Transcoder OR RSV
```

For example, the content server only has content in WML format (`text/vnd.wap.wml`). The Mobile Data Service finds this entry for this MIME type:

```
text/vnd.wap.wml : vnd.wap.wml
```

According to this entry, the Mobile Data Service must apply the `vnd.wap.wml` transcoder to any content in WML format.

Default entry

A default entry specifies the default action that the Mobile Data Service performs if no entry is found for a particular MIME type:

```
default : Transcoder or RSV
```

For example, content is forwarded to the handheld without change if no entry is found for its content type in the following default entry:

```
default:pass
```

Creating transcoders

Convert HTML tags and content to uppercase

Follow transcoder package hierarchy

Define the HTTP content transcoder in a package that is called `net.rim.protocol.http.content.transcoder.<`*`transcoder name`*`>`. All HTTP content transcoders must be defined in the `Transcoder` package hierarchy.

```
package net.rim.protocol.http.content.transcoder.uppercasehtml;
```

Extend HTTPContentTranscoder

The class name for a transcoder must be `Transcoder`. Transcoders extend the `HttpContentTranscoder` class.

In this example, the HTTP content transcoder is defined in a public class named `Transcoder`.

```
public class Transcoder extends HttpContentTranscoder { ... }
```

Define HTTP headers

Define constants for HTTP headers and the strings that the transcoder adds to these headers.

```
private static final String CONTENTTYPE_HEADER_NAME = "Content-Type";
private static final String CONTENTLENGTH_HEADER_NAME = "Content-Length";
private static final String ACCEPT_HEADER_NAME = "Accept";
// This line is added to the Accept header field if it exists when
// the handheld issues an HTTP request.
private static final String ACCEPTLINE= "text/html";
// This line identifies the output content type this transcoder produces
private static final String OUTPUT_TYPE= "text/UPPERCASEHTML";
```

Create a mapping of input and output types

Implement `getMapOfOutputToAcceptLine()` to create a mapping of the transcoder's possible input and output types.

The Mobile Data Service collects this information when it starts so that it can modify the `Accept` header field before it sends an HTTP request to an HTTP server.

```
public HashMap getMapOfOutputToAcceptLine() {
   HashMap mapping = new HashMap();
   mapping.put(OUTPUT_TYPE, ACCEPTLINE);
   return mapping;
}
```

Set the connection URL

Define a method to set the connection URL when the handheld requests an open HTTP connection.

```
public void setURL(URL newURL) {
   url = newURL;
}
```

Define handheld request processing

Implement `transcodeDevice(HttpRequest request)` to define any processing on the handheld request before the Mobile Data Service forwards it to the destination server. In the following example, no processing is required.

A handheld application can request a specific transcoder by using the HTTP header field that is called `Content-Transcoder`.

```
public void transcodeDevice(HttpRequest request) throws
   HttpContentTranscoderException {
   // implementation
}
```

Define handheld response processing

Implement `transcodeDevice(HttpResponse response)` to define any processing on the handheld response before the Mobile Data Service forwards it to the destination server. In the following example, no processing is required.

```
public void transcodeDevice(HttpResponse response) throws
   HttpContentTranscoderException {
   // implementation
}
```

Define server request processing

Implement `transcodeServer(HttpRequest request)` to define any processing on the server request before the Mobile Data Service forwards it to the handheld.

```
public void transcodeServer(HttpRequest request) throws
   HttpContentTranscoderException {
   try {
   // retrieve the request content, which, in this case, is in HTML
      byte[] requestContent = request.getContent();
      if (requestContent != null) {
         // convert the content to String object
         String requestContentAsString = new
         String(requestContent).toUpperCase();
         // convert the requestContentAsString to bytes again
```

```
            requestContent = requestContentAsString.getBytes();
        }
        else {
            // server is not responding with appropriate content
            // send an HTML message to indicate this.
            StringBuffer sb = new StringBuffer();
            sb.append("<HTML>\n");
            sb.append("<HEAD>\n");
            sb.append("<TITLE> UPPERCASEHTML TRANSCODER</title>\n");
            sb.append("</HEAD>\n");
            sb.append("<BODY>\n");
            sb.append("SERVER IS NOT PUSHING APPROPRIATE CONTENT\n");
            sb.append("</BODY\n");
            sb.append("</HTML>\n");
            requestContent = sb.toString().getBytes();
        }
        request.setContent(requestContent);
        // update the Content-Length
        HttpHeader contentLengthHeader =
        request.getHeader(CONTENTLENGTH_HEADER_NAME);
        if (contentLengthHeader != null) {
            contentLengthHeader.setValue("" + requestContent.length);
        }
        else {
            // The server did not send the Content-Length.
            // No update is needed.
        }
        // update the Content-Type
        HttpHeader contentTypeHeader =
        request.getHeader(CONTENTTYPE_HEADER_NAME);
        if (contentTypeHeader != null) {
            contentTypeHeader.setValue(OUTPUT_TYPE);
        }
        else {
            // add the Content Type here if the server does not specify one
            request.putHeader(new HttpHeader(
            CONTENTTYPE_HEADER_NAME, OUTPUT_TYPE));
        }
    } catch (Throwable t) {
        throw new HttpContentTranscoderException(t.toString());
    }
}
```

Define server response processing

Implement `transcodeServer(HttpResponse response)` to define any processing on a server response that includes content from the server before the Mobile Data Service forwards it to the handheld. If no content is attached to the response, the Mobile Data Service forwards the response to the handheld without any changes.

```
public void transcodeServer(HttpResponse response) throws
HttpContentTranscoderException {
    try {
        // retrieve the response content, which in this case is in HTML
        byte[] responseContent = response.getContent();
        if (responseContent != null) {
                // convert the content to String object
                String responseContentAsString = new
                String(responseContent).toUpperCase();
                // convert the responseContentAsString to bytes again
                responseContent = responseContentAsString.getBytes();
            }
            else {
                //  no response is received from the server
                StringBuffer sb = new StringBuffer();
                sb.append("<HTML>\n");
                sb.append("<HEAD>\n");
                sb.append("<TITLE> UPPERCASEHTML TRANSCODER </title>\n");
                sb.append("</HEAD>\n");
                sb.append("<BODY>\n");
                sb.append("SERVER IS NOT RESPONDING\n");
                sb.append("</BODY\n");
                sb.append("</HTML>\n");
                responseContent = sb.toString().getBytes();
            }
            response.setContent(responseContent);
            // update the Content-Length
            HttpHeader contentLengthHeader =
            response.getHeader(CONTENTLENGTH_HEADER_NAME);
            if (contentLengthHeader != null) {
                contentLengthHeader.setValue("" + responseContent.length);
            }
            else {
                // server did not send Content-Length so no update is required
            }
            // update the Content-Type
            HttpHeader contentTypeHeader =
            response.getHeader(CONTENTTYPE_HEADER_NAME);
            if (contentTypeHeader != null) {
                contentTypeHeader.setValue(OUTPUT_TYPE);
            }
            else {
                // add the Content Type here
                response.putHeader(new
                HttpHeader(CONTENTTYPE_HEADER_NAME, OUTPUT_TYPE));
            }
        } catch (Throwable t) {
        throw new HttpContentTranscoderException(t.toString());
        }
        }
}
```

Compile and install transcoders

1. Compile the transcoder class files, including bmds.jar in the class path.
   ```
   javac -classpath ".;MDS_home\samples\bmds.jar" Transcoder.java
   ```
2. Create a .jar file for the transcoder.
3. Install the transcoder .jar file in one of the following ways:

- **BlackBerry Enterprise Server**: Add the transcoder .jar file into the lib\ext folder of the JRE. The .jar file must be accessible to the VM.
- **Simulator**: Add the transcoder .jar file to the `MDS_home\classpath` folder in the BlackBerry JDE.

4. Open the MDS_home\config\httpcontenttranscoderslist.property file.
5. Add one or more entries to specify when the transcoder should be used. For example, to specify that the `uppercasehtml` transcoder should be used to map HTML content to uppercase HTML, add this entry:
   ```
   text/html -> text/UPPERCASEHTML : uppercasehtml
   ```
6. Save the property file.
7. Restart the Mobile Data Service.

Glossary

A

ALX	Application Loader XML
API	application programming interface
APN	Access Point Name

C

CA	Certificate Authority
CDMA	Code Division Multiple Access
CHAP	Challenge Handshake Authentication Protocol
cHTML	Compact Hypertext Markup Language
CLDC	Connected Limited Device Configuration
CPU	central processing unit

D

DES	Data Encryption Standard
DNS	Domain Name System

G

GIF	Graphics Interchange Format
GPRS	General Packet Radio Service
GUI	graphical user interface
GUID	globally unique identifier

H

HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer

I

i18n	internationalization
IDE	integrated development environment
iDEN	Integrated Digital Enhanced Network
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
I/O	input/output
IP	Internet Protocol
IPPP	IP Proxy Protocol
ISDN	Integrated Services Digital Network

J

J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
JAD	Java Application Descriptor
JAR	Java Archive
JDE	Java Development Environment
JPEG	Joint Photographic Experts Group
JRE	Java Runtime Environment

K

KB	kilobytes
KVM	Kilobyte virtual machine

L

LAN	local area network
LDAP	Lightweight Directory Access Protocol
LTPA	Lightweight Third-Party Authentication

M

MB	megabyte
MHz	megahertz
MIDlet	MIDP application
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions
MSISDN	Mobile Station ISDN

O

OCSP	Online Certificate Status Protocol

P

PAP	Password Authentication Protocol
PDA	personal digital assistant
PIM	personal information management
PIN	personal identification number
PNG	Portable Network Graphics

R

RAM	random access memory
RRC	Radio Resource Control
RTC	real-time clock

S

SDK	software development kit
SIM	Subscriber Identity Module
SMS	Short Message Service

SRAM
: static random access memory

SRP
: Service Relay Protocol

SSL
: Secure Sockets Layer

T

TCP
: Transmission Control Protocol

TCP/IP
: Transmission Control Protocol/Internet Protocol

TIFF
: Tag Image File Format

TLS
: Transport Layer Security

U

UDP
: User Datagram Protocol

UI
: user interface

URI
: Uniform Resource Identifier

URL
: Uniform Resource Locator

UTC
: Universal Time Coordinate

V

VM
: virtual machine

W

WAP
: Wireless Application Protocol

WBMP
: wireless bitmap

WML
: Wireless Markup Language

WMLC
: Wireless Markup Language Compiled

WTLS
: Wireless Transport Layer Security

X

XHTML
: Extensible Hypertext Markup Language

XML
: Extensible Markup Language

Index

©2004 Research In Motion Limited
Published in Canada.