

BlackBerry Java Development Environment

Version: 4.7.0

Development Guide

Contents

1	Creating user interfaces.....	7
	Screens.....	7
	Screen classes.....	7
	Create a screen.....	8
	How the BlackBerry JVM manages screens.....	8
	Providing screen navigation when using a MainScreen.....	9
	Manage a drawing area.....	9
	Touch screen orientation and direction.....	10
	Retrieve the orientation of the touch screen.....	10
	Restrict the touch screen direction.....	11
	Receive notification when the size of the drawable area of the touch screen changes.....	11
	Receive notification when the orientation of the touch screen changes.....	11
	Working with the accelerometer of a BlackBerry device.....	12
	Types of accelerometer data.....	12
	Accelerometer.....	12
	Retrieve accelerometer data at specific intervals.....	13
	Query the accelerometer when the application is in the foreground.....	13
	Query the accelerometer when the application is in the background.....	14
	Store accelerometer readings in a buffer.....	14
	Retrieve accelerometer readings from a buffer.....	15
	Get the time a reading was taken from the accelerometer.....	15
	UI components.....	16
	Add a UI component to a screen.....	16
	Create a dialog box.....	16
	Create a bitmap.....	16
	Create a button.....	16
	Create a list.....	17
	Create a drop-down list.....	17
	Create a search field.....	17
	Create a check box.....	19
	Create an option button.....	19
	Create a date field.....	20
	Creating a text field.....	20

Create a progress indicator.....	21
Create a text label.....	21
Create a list box.....	21
Create a field to display a tree view.....	22
Add a UI component to a screen.....	23
Create a custom field.....	23
Add a menu item to a BlackBerry Device Software application.....	27
Adding a menu item to a BlackBerry Device Software application.....	28
Register a menu item.....	28
Arrange UI components.....	29
Create a layout manager.....	30
Aligning a field to a line of text.....	31
Events.....	31
Respond to navigation events.....	31
Determine the type of input method.....	31
Respond to BlackBerry device user interaction.....	32
Touch screen events.....	32
Types of touch screen events.....	32
Respond to touch screen events.....	34
Keyboard on a BlackBerry device with a touch screen.....	41
Change the state of the keyboard on a BlackBerry device with a touch screen.....	41
Display the keyboard on a BlackBerry device with a touch screen.....	42
Hide the keyboard on a BlackBerry device with a touch screen.....	42
Spell check.....	43
Add spell check functionality.....	43
Listen for a spell check event.....	44
Accessibility.....	45
Integrating with assistive technology software.....	45
Notifying an assistive technology application when the UI changes.....	47
UI changes that trigger a notification to an assistive technology application.....	47
UI component states and properties.....	47
Provide an assistive technology application with information about a UI change.....	48
Provide an assistive technology application with information about text changes.....	50
Provide an assistive technology application with access to information from a table.....	52
Provide an assistive technology application with access to numeric values.....	53

Enable an assistive technology application to receive notification of UI events.....	54
2 Storing data.....	56
Data management.....	56
Support for APIs to store data to persistent memory.....	56
Data management.....	56
Access to memory.....	56
Persistent data storage.....	57
Create a persistent data store.....	57
Store persistent data.....	57
Retrieve persistent data.....	58
Remove persistent data.....	58
MIDP record storage.....	59
Create an MIDP record store.....	59
Add a record to a record store.....	59
Retrieve a record from a record store.....	59
Retrieve all records from a record store.....	59
Collections.....	60
Retrieve a collection from persistent storage.....	60
Create a collection listener to notify the system when a collection changes.....	61
Remove a collection listener that notifies the system when a collection changes.....	61
Notify the system when a collection changes.....	62
Runtime storage.....	62
Retrieve the runtime store.....	62
Add an object in the runtime store.....	63
Replace an object in the runtime store.....	63
Retrieve a registered runtime object.....	63
Retrieve an unregistered runtime object.....	64
3 Creating connections.....	65
Network gateways.....	65
Using the BlackBerry Enterprise Server as an intranet gateway.....	65
Using the wireless service provider's Internet gateway.....	65
Retrieve the wireless network name.....	65
Connections.....	66
Use HTTP authentication.....	66

Use an HTTPS connection.....	67
Use a socket connection.....	68
Use a datagram connection.....	69
Use a USB or serial port connection.....	71
Use a Bluetooth serial port connection.....	72
Wi-Fi connections.....	73
Wireless access families.....	73
Retrieve the wireless access families that a BlackBerry device supports.....	73
Determine if a BlackBerry device supports multiple wireless access families.....	74
Determine the wireless access family transceivers that are turned on.....	74
Turn on the transceiver for a wireless access family.....	74
Turn off the transceiver for a wireless access family.....	74
Check if the Wi-Fi transceiver is turned on.....	74
Check if the Wi-Fi transceiver is connected to a wireless access point.....	74
Retrieve the status of the wireless access point or the active Wi-Fi profile.....	75
Open a Wi-Fi socket connection.....	75
Open a Wi-Fi HTTP connection.....	75
Open a Wi-Fi HTTPS connection.....	76
4 Managing applications.....	77
Application manager.....	77
Retrieve information about a BlackBerry Java Application.....	77
Communicate with another BlackBerry Java Application.....	77
Determine the services that are available to a BlackBerry Java Application.....	78
Application control.....	78
Allow a BlackBerry device application to request access to resources.....	78
Code modules.....	79
Retrieve module information.....	79
Access control messages.....	79
Displaying a message for an operation that requires user permission.....	79
Display an application control message to a user.....	80
5 Using custom messages and folders in the message list.....	81
Creating a module for background processes.....	81
Creating a module for the UI.....	81
Create the module for background processes.....	81

Start the module for background processes or the module for the UI.....	82
Create an icon for a custom message.....	82
Create a custom folder in the message list.....	83
Send a notification when a custom folder changes.....	84
Create an indicator for the number of messages in a custom folder.....	85
Hide an indicator for a custom folder.....	86
Remove an indicator for a custom folder.....	86
6 Applications for push content.....	87
Types of push applications.....	87
Types of push requests.....	87
7 Localizing BlackBerry device applications.....	89
Multilanguage support.....	89
Files required for localization.....	89
Manage localization files for a suite of BlackBerry device applications.....	90
8 Controlling access to APIs and application data.....	92
Check if a code signature is required.....	92
Java APIs with controlled access.....	92
Register to use controlled APIs.....	93
Restrictions on code signatures.....	93
Request a code signature.....	94
Register a signature key using a proxy server.....	94
Sign an application using a proxy server.....	95
View the signature status for an application.....	95
Using keys to protect APIs and data.....	95
Protect APIs using code signing keys.....	96
Protect runtime store data using code signing keys.....	96
Protect persistent data using code signing keys.....	97
9 Testing a BlackBerry device application.....	98
Testing applications on a BlackBerry Smartphone Simulator.....	98
Testing applications on a BlackBerry device.....	98
Testing applications using the compiled .cod files.....	98
Install and remove a .cod file for testing.....	99

Save a .cod file from a device to a computer.....	99
Retrieve information about a .cod file.....	99
10 Packaging and distributing a BlackBerry Java Application.....	101
Preverify a BlackBerry device application.....	101
Application distribution over the wireless network.....	101
Wireless pull (user-initiated).....	101
Wireless push (server-initiated).....	101
Distributing BlackBerry Java Applications over the wireless network.....	101
Extract sibling .cod files.....	101
Modifying information for a MIDlet suite.....	102
Correct the .cod file sizes listed in a .jad file.....	103
Create .jad files that reference multiple .cod files.....	103
Distributing BlackBerry device applications with the BlackBerry Desktop Software.....	104
Elements in BlackBerry device application .alx file.....	104
Properties of BlackBerry device application .jad files.....	109
Application distribution through a computer connection.....	110
Distribute an application from a computer.....	110
Distribute an application from a web page.....	110
Distribute an application for testing.....	110
Distributing an application from a computer.....	110
Create an application loader file.....	110
Install a BlackBerry device application on a specific device.....	111
Specifying supported versions of the BlackBerry Device Software.....	111
11 Glossary.....	113
12 Provide feedback.....	116
13 Legal notice.....	117

Creating user interfaces

1

Screens

The main structure for a BlackBerry® device user interface is the `Screen` object. A BlackBerry device application can have multiple screens open at the same time, but only one screen can be displayed at a time.

The BlackBerry® Java® Virtual Machine maintains `Screen` objects in a display stack. The display stack is an ordered set of `Screen` objects. The screen at the top of the stack is the active screen that the BlackBerry device user sees. When a BlackBerry device application displays a screen, it pushes the screen to the top of the stack. When a BlackBerry device application closes a screen, it removes the screen off the top of the stack and displays the next screen on the stack, redrawing it as necessary. Each screen can appear only once in the display stack. The BlackBerry JVM throws a runtime exception if a `Screen` that the BlackBerry device application pushes to the stack already exists. A BlackBerry device application must remove screens from the display stack when the BlackBerry device user finishes interacting with them so that the BlackBerry device application uses memory efficiently. Use only a few modal screens at one time, because each screen uses a separate thread.

The UI APIs initialize simple `Screen` objects. Once you create a screen, you can add fields and a menu to the screen, and display the screen to the BlackBerry device user by pushing it on to the display stack. The `Menu` object has associated menu items that are runnable objects that perform a specific task when the BlackBerry device user selects one of the items. For example, you can use menu items to invoke the necessary code to establish a network connection, commit a data object to memory, or close a BlackBerry device application. You can customize the BlackBerry device UI and implement new field types, as required. You can also add custom navigation.

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

For knowledge base articles about displaying and working with screens, visit the BlackBerry Developer Zone at www.blackberry.com/developers.

Screen classes

Class	Description
<code>Screen</code>	Use the <code>Screen</code> class to define a manager to lay out UI components on the screen and to define a specific type of screen using the styles that the constants on the <code>Field</code> superclass define.

Class	Description
<code>FullScreen</code>	Use a <code>FullScreen</code> class to provide an empty screen that you can add UI components to in a standard vertical layout. If you want to use another layout style, such as horizontal or diagonal, use a <code>Screen</code> class instead and add a <code>Manager</code> to it.
<code>MainScreen</code>	<p>Use a <code>MainScreen</code> class to create a screen with the following standard UI components:</p> <ul style="list-style-type: none"> • default position of a screen title, with a <code>SeparatorField</code> after the title • main scrollable section contained in a <code>VerticalFieldManager</code> • default menu with a Close menu item • default close action when the BlackBerry® device user clicks the Close menu item or presses the Escape key <p>You should consider using a <code>MainScreen</code> object for the first screen of your BlackBerry device application to maintain consistency with other BlackBerry device applications.</p>

Create a screen

1. Import the following classes:
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.container.FullScreen`
 - `net.rim.device.api.ui.container.MainScreen`
2. Extend the `Screen` class or one of its subclasses, `FullScreen` or `MainScreen`.

How the BlackBerry JVM manages screens

The BlackBerry® Java® Virtual Machine maintains `Screen` objects in a display stack, which is an ordered set of `Screen` objects. The screen at the top of the stack is the active screen that the BlackBerry device user sees. When a BlackBerry device application displays a screen, it pushes the screen to the top of the stack. When a BlackBerry device application closes a screen, it removes the screen off the top of the stack and displays the next screen on the stack, redrawing it as necessary. Each screen can appear only once in the display stack. The BlackBerry JVM throws a runtime exception if a `Screen` that the BlackBerry device application pushes to the stack already exists. A BlackBerry device application must remove screens from the display stack when the BlackBerry device user finishes interacting with them so that the BlackBerry device application uses memory efficiently. Use only a few modal screens at one time, because each screen uses a separate thread.

Providing screen navigation when using a MainScreen

A `MainScreen` object provides default navigation to your BlackBerry® device application. Avoid using buttons or other UI elements that take up space on the screen.

```
MainScreen mainScreen = new MainScreen(DEFAULT_MENU | DEFAULT_CLOSE);
```

Manage a drawing area

The `Graphics` object represents the entire drawing surface that is available to the BlackBerry® device application. To limit this area, divide it into `XYRect` objects. Each `XYPoint` represents a point on the screen, which is composed of an X co-ordinate and a Y co-ordinate.

1. Import the following classes:
 - `net.rim.device.api.ui.Graphics`
 - `net.rim.device.api.ui.XYRect`
 - `net.rim.device.api.ui.XYPoint`
2. Create `XYPoint` objects, to represent the top left and bottom right points of a rectangle. Create an `XYRect` object using the `XYPoint` objects, to create a rectangular clipping region.

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

3. Invoke `Graphics.pushContext()` to make drawing calls that specify that the region origin should not adjust the drawing offset. Invoke `Graphics.pushContext()` to push the rectangular clipping region to the context stack. Invoke `Graphics.drawRect()` to draw a rectangle, and invoke `Graphics.fillRect()` to fill the rectangle. Invoke `Graphics.popContext()` to pop the current context off of the context stack.

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

4. Invoke `pushRegion()` and specify that the region origin should adjust the drawing offset. Invoke `Graphics.drawRect()` to draw a rectangle and invoke `Graphics.fillRect()` to fill a rectangle. Invoke `Graphics.popContext()` to pop the current context off of the context stack.

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

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

```
graphics.pushContext(rectangle);
graphics.invert(rectangle);
graphics.popContext();
```

7. Invoke `translate()`. 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 clips it.

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

Touch screen orientation and direction

You can specify the orientation and direction of the `Screen` in a BlackBerry® device application.

Orientation refers to the `Screen` aspect ratio, relative to the ground, where the left, right, top, bottom, front, or back of the device are furthest from the ground.

Direction refers to the top of the drawing area of the screen, relative to the location of the BlackBerry logo. If you change the direction that a `Screen` supports, the change takes effect the next time the `Screen` orientation of the device changes.

A `Screen` for a BlackBerry device application can have the following directions and orientations:

Direction

- `DIRECTION_NORTH` - the top of the drawable area is the screen side closest to the BlackBerry logo
- `DIRECTION_WEST` - the top of the drawable area is to the left of the logo
- `DIRECTION_EAST` - the top of the drawable area is to the right of the logo
- `DIRECTION_SOUTH` - the top of the drawable area is the screen side farthest from the BlackBerry logo

Orientation

- `DIRECTION_LANDSCAPE` - the top of the drawable area is the screen side farthest from the BlackBerry logo
- `DIRECTION_PORTRAIT` - the screen is display in portrait view

Retrieve the orientation of the touch screen

1. Import the `net.rim.device.api.system.Display` class.
2. Invoke `net.rim.device.api.system.Display.getOrientation()`.

```
int orientation = Display.getOrientation();
```

Restrict the touch screen direction

1. Import the following classes:
 - `net.rim.device.api.ui.Ui`
 - `net.rim.device.api.ui.UiEngineInstance`
2. Invoke `net.rim.device.api.ui.Ui.getUiEngineInstance()`.

```
UiEngineInstance _ue;  
_ue = Ui.getUiEngineInstance();
```

3. Invoke `net.rim.device.api.ui.UiEngineInstance.setAcceptableDirections(byte flags)` and pass the argument for the direction of the Screen.

```
_ue.setAcceptableDirections(Display.DIRECTION_WEST);
```

Receive notification when the size of the drawable area of the touch screen changes

1. Import the following classes:
 - `javax.microedition.lcdui.Canvas`
 - `net.rim.device.api.ui.component.Dialog`
2. Extend `javax.microedition.lcdui.Canvas`.
3. Override `Canvas.sizeChanged(int, int)`.

```
protected void sizeChanged(int w, int h) {  
    Dialog.alert("The size of the Canvas has changed");  
}
```

Receive notification when the orientation of the touch screen changes

1. Import the `net.rim.device.api.ui.Manager` class.
2. Override `net.rim.device.api.ui.Manager.sublayout(int, int)`.

```
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);
}
```

Working with the accelerometer of a BlackBerry device

On a BlackBerry® device with an accelerometer, you can create a BlackBerry device application that listens for accelerometer events and retrieves data about the orientation of the BlackBerry device.

The BlackBerry OS converts accelerometer readings into orientation coordinates such as `TOP_UP`, `FACE_UP`, and `LEFT_UP`. The BlackBerry OS may also rotate the screen in response to accelerometer readings.

Querying the accelerometer consumes battery power. Connections to the accelerometer should not be kept open for long periods of time.

Types of accelerometer data

A BlackBerry® device application can retrieve data from the accelerometer.

Data type	Description
orientation	The orientation of the BlackBerry device with respect to the ground.
acceleration	The acceleration of the rotation of the BlackBerry device.

For more information about types of data from the accelerometer, see the API reference for the BlackBerry® Java® Development Environment.

Accelerometer

A BlackBerry® device with a touch screen can include an accelerometer, which is designed to sense the orientation and acceleration of the BlackBerry device. When a BlackBerry device user moves the BlackBerry device, the accelerometer can sense the movement in 3-D space along the x, y, and z axis. A BlackBerry device user can change the orientation of the device which can change the display direction of a screen for a BlackBerry device application between portrait and landscape.

You can use the Accelerometer APIs in the `net.rim.device.api.system` package to respond to the orientation and acceleration of a BlackBerry device. For example, you can manipulate a game application to change the direction and speed of a moving object on the screen as a user moves and rotates the BlackBerry device at different speeds.

Retrieve accelerometer data at specific intervals

If a BlackBerry® device application opens a channel to the accelerometer when the application is in the foreground, when the application is in the background, the channel pauses and the accelerometer is not queried. If a BlackBerry device application invokes `AccelerometerSensor.Channel.getLastAccelerationData(short[])` at close intervals or when the BlackBerry device is not in motion, the method might return duplicate values.

1. Import the following classes:
 - `net.rim.device.api.system.AccelerometerSensor.Channel`;
 - `net.rim.device.api.system.AccelerometerSensor`;
2. Open a channel to the accelerometer. While a channel is open, the BlackBerry device application will query the accelerometer for information.

```
Channel rawDataChannel = AccelerometerSensor.openRawDataChannel  
( Application.getApplication() );
```

3. Create an array to store the accelerometer data.

```
short[] xyz = new short[ 3 ];
```

4. Create a thread.

```
while( running ) {
```

5. Invoke `Channel.getLastAccelerationData(short[])` to retrieve data from the accelerometer.

```
rawDataChannel.getLastAccelerationData( xyz );
```

6. Invoke `Thread.sleep()` to specify the interval between queries to the accelerometer, in milliseconds.

```
Thread.sleep( 500 );
```

7. Invoke `Channel.close()` to close the channel to the accelerometer.

```
rawDataChannel.close();
```

Query the accelerometer when the application is in the foreground

1. Import the following classes:
 - `net.rim.device.api.system.AccelerometerChannelConfig`
 - `net.rim.device.api.system.AccelerometerSensor.Channel`
2. Open a channel to retrieve acceleration data from the accelerometer.

```
Channel channel = AccelerometerSensor.openRawAccelerationChannel  
( Application.getApplication());
```

3. Invoke `Thread.sleep()` to specify the interval between queries to the accelerometer, in milliseconds.

```
short[] xyz = new short[ 3 ];
while( running ) {
    channel.getLastAccelerationData( xyz );
    Thread.sleep( 500 );
}
```

4. Invoke `Channel.close()` to close the channel to the accelerometer.

```
channel.close();
```

Query the accelerometer when the application is in the background

1. Import the following classes:
 - `net.rim.device.api.system.AccelerometerChannelConfig`;
 - `net.rim.device.api.system.AccelerometerSensor.Channel`;
2. Create a configuration for a channel to the accelerometer.
3. Invoke `AccelerometerChannelConfig.setBackgroundMode(Boolean)`, to specify support for an application that is in the background.

```
AccelerometerChannelConfig channelConfig = new AccelerometerChannelConfig
( AccelerometerChannelConfig.TYPE_RAW );
```

4. Invoke `AccelerometerSensor.openChannel()`, to open a background channel to the accelerometer.
5. Invoke `Thread.sleep()` to specify the interval between queries to the accelerometer, in milliseconds.

```
channelConfig.setBackgroundMode( true );
```

```
Channel channel = AccelerometerSensor.openChannel( Application.getApplication(),
channelConfig );
```

```
short[] xyz = new short[ 3 ];
while( running ) {
    channel.getLastAccelerationData( xyz );
    Thread.sleep( 500 );
}
```

6. Invoke `Channel.close()` to close the channel to the accelerometer.

```
channel.close();
```

Store accelerometer readings in a buffer

1. Import the following classes:
 - `net.rim.device.api.system.AccelerometerChannelConfig`;
 - `net.rim.device.api.system.AccelerometerSensor.Channel`;
2. Create a configuration for a channel to the accelerometer.

```
AccelerometerChannelConfig channelConfig = new AccelerometerChannelConfig
( AccelerometerChannelConfig.TYPE_RAW );
```

3. Invoke `AccelerometerChannelConfig.setSamplesCount()`, to specify the number of acceleration readings to store in the buffer. Each element in the buffer contains acceleration readings for the x, y, and z axes and data on when the reading took place.

```
channelConfig.setSamplesCount( 500 );
```

4. Invoke `AccelerometerSensor.openChannel()` to open a channel to the accelerometer.

```
Channel bufferedChannel = AccelerometerSensor.openChannel
( Application.getApplication(), channelConfig );
```

Retrieve accelerometer readings from a buffer

1. Import the following classes:
 - `net.rim.device.api.system.AccelerometerData`;
 - `net.rim.device.api.system.AccelerometerSensor.Channel`;
2. Query the buffer for accelerometer data.
3. Invoke `AccelerometerData.getNewBatchLength()`, to get the number of readings retrieved since the last query.
4. Invoke `AccelerometerData.getXAccHistory()`, `AccelerometerData.getYAccHistory()`, and `AccelerometerData.getZAccHistory()` to retrieve accelerometer data from the buffer for each axis.

```
short[] xAccel = accData.getXAccHistory();
short[] yAccel = accData.getYAccHistory();
short[] zAccel = accData.getZAccHistory();
```

Get the time a reading was taken from the accelerometer

1. Import the `net.rim.device.api.system.AccelerometerData` class.
2. Query the buffer for accelerometer data.

```
AccelerometerData accData;
accData = bufferedChannel.getAccelerometerData();
```

3. Invoke `AccelerometerData.getSampleTsHistory()`.

```
long[] queryTimestamps = accData.getSampleTsHistory();
```

UI components

Add a UI component to a screen

1. Import the following classes:
 - `net.rim.device.api.ui.component.CheckboxField`
 - `net.rim.device.api.ui.container.MainScreen`
2. Create an instance of a UI component.

```
CheckboxField myCheckbox = new CheckboxField("First checkbox", true);
```

3. Add the UI component to your extension of a `Screen` class.

```
mainScreen.add(myCheckbox);
```

Create a dialog box

Use alert dialog boxes to notify users of a critical action such as turning off the BlackBerry® device or an error such as typing information that is not valid. An exclamation point (!) indicator appears in an alert dialog box. To close an alert dialog box, users can click OK or press the Escape key. For more information on other types of dialog boxes, see the API reference for BlackBerry® Java® Development Environment.

1. Import the `net.rim.device.api.ui.component.Dialog` class.
2. Create an alert dialog box specifying the alert text that you want to display.

```
Dialog.alert("Specify the alert text that you want to display.")
```

Create a bitmap

1. Import the `net.rim.device.api.ui.component.BitmapField` class.
2. Create an instance of a `BitmapField`.

```
BitmapField myBitmapField = new BitmapField();
```

Create a button

1. Import the `net.rim.device.api.ui.component.ButtonField` class.
2. Create an instance of a `ButtonField` using a style parameter.

```
ButtonField mySubmitButton = new ButtonField("Submit");
```

Create a list

Create a drop-down list

Use drop-down lists to provide a set of mutually exclusive values.

1. Import the following classes:
 - `java.lang.String`
 - `net.rim.device.api.ui.component.ObjectChoiceField`
 - `net.rim.device.api.ui.container.MainScreen`
2. Create an instance of an `ObjectChoiceField`, providing an object array as a parameter to create a drop-down list that contains objects.

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

Create a search field

You can create an application that uses the `KeywordFilterField` class, included in the `net.rim.device.api.ui.component` package, to provide a UI field that consists of a single text input field and a list of selectable elements. As users type text in a search field, the application filters the elements in the list that begin with the search text. For more information about using the `KeywordFilterField` class, see the `KeywordFilterFieldDemo` sample application, included with the BlackBerry® Java® Development Environment version 4.3.1 or later.

1. Import the following classes:
 - `net.rim.device.api.ui.component.KeywordFilterField`
 - `net.rim.device.api.collection.util.SortedReadableList`
 - `java.util.Vector`
 - `java.io.InputStream`
 - `net.rim.device.api.io.LineReader`
 - `java.lang.String`
2. Import the `net.rim.device.api.ui.component.KeywordProvider` interface.
3. Create variables. In the following code sample, `CountryList` extends the `SortedReadableList` class and implements the `KeywordProvider` interface.

```
private KeywordFilterField _keywordField;
private CountryList _countryList;
private Vector _countries;
```

4. To create a list of selectable text items, populate a vector with data from a text file.

```
_countries = getDataFromFile();
```

5. Create an instance of a class that extends the `SortedReadableList` class.

```
_CountryList = new CountryList(StringComparator.getInstance(true), _countries);
```

6. To specify the elements of the list, create a new instance of a `KeywordFilterField` object.

```
_keywordField = new KeywordFilterField();
```

7. Invoke `KeywordFilterField.setList()`.

```
_keywordField.setList(_CountryList, _CountryList);
```

8. Set a label for the input field of the `KeywordFilterField`.

```
_keywordField.setLabel("Search: ");
```

9. Create the main screen of the application and add a `KeywordFilterField` to the main screen.

```
KeywordFilterDemoScreen screen = new KeywordFilterDemoScreen(this, _keywordField);
screen.add(_keywordField.getKeywordField());
screen.add(_keywordField);
pushScreen(screen);
```

10. To create a method that populates and returns a vector of `Country` objects containing data from text file, in the method signature, specify `Vector` as the return type.

```
public Vector getDataFromFile()
```

11. Create and store a reference to a new `Vector` object.

```
Vector countries = new Vector();
```

12. Create an input stream to the text file.

```
InputStream stream = getClass().getResourceAsStream("/Data/CountryData.txt");
```

13. Read CRLF delimited lines from the input stream.

```
LineReader lineReader = new LineReader(stream);
```

14. Read data from the input stream one line at a time until you reach the end of file flag. Each line is parsed to extract data that is used to construct `Country` objects.

```
for(;;){
    //Obtain a line of text from the text file
    String line = new String(lineReader.readLine());

    //If we are not at the end of the file, parse the line of text
    if(!line.equals("EOF")) {
        int space1 = line.indexOf(" ");
        String country = line.substring(0,space1);
        int space2 = line.indexOf(" ",space1+1);
        String population = line.substring(space1+1,space2);
        String capital = line.substring(space2+1,line.length());
```

```

        // Create a new Country object
        countries.addElement(new Country
(country,population,capital));
    }
    else {
        break;
    }
} // end the for loop
return countries;

```

- To add a keyword to the list of selectable text items, invoke `SortedReadableList.doAdd(element)`.

```
SortedReadableList.doAdd(((Country)countries.elementAt(i)).getCountryName());
```

- To update the list of selectable text items, invoke `KeywordFilterField.updateList()`.

```
_keywordField.updateList();
```

- To obtain the key word that a BlackBerry device user typed into the `KeywordFilterField`, invoke `KeywordFilterField.getKeyword()`.

```
String userTypedWord = _keywordField.getKeyword();
```

Create a check box

- Import the `net.rim.device.api.ui.component.CheckboxField` class.
- Create an instance of a `CheckboxField`.

```
CheckboxField myCheckbox = new CheckboxField("First checkbox", true);
```

Create an option button

Use option buttons to indicate a set of mutually exclusive but related choices.

- Import the following classes:
 - `net.rim.device.api.ui.component.RadioButtonGroup`
 - `net.rim.device.api.ui.component.RadioButtonField`
- Create an instance of a `RadioButtonGroup`.
- Create an instance of a `RadioButtonField` for each option you want to make available to the BlackBerry® device user.
- Invoke `RadioButtonGroup.add()` to add the `RadioButtonField` instances to the `RadioButtonGroup`.

```
RadioButtonGroup rbGroup = new RadioButtonGroup();
```

```
RadioButtonField rbField = new RadioButtonField("First option");
RadioButtonField rbField2 = new RadioButtonField("Second option");
```

```
rbGroup.add(rbField);
rbGroup.add(rbField2);
```

Create a date field

1. Import the `net.rim.device.api.ui.component.DateField` class.
2. Create an instance of a `DateField`, providing the value returned by `System.currentTimeMillis()` as a parameter to return the current time.

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

Creating a text field

Create a read-only text field that allows formatting

1. Import the `net.rim.device.api.ui.component.RichTextField` class.
2. Create an instance of a `RichTextField`.

```
RichTextFielddrich=newRichTextField("RichTextField");
```

Create an editable text field that has no formatting and accepts filters

1. Import the following classes:
 - `net.rim.device.api.ui.component.BasicEditField`
 - `net.rim.device.api.ui.component.EditField`
2. Create an instance of a `BasicEditField`.

```
BasicEditFielddbfbf=newBasicEditField("BasicEditField: ", "",
10, EditField.FILTER_UPPERCASE);
```

Create an editable text field that allows special characters

1. Import the `net.rim.device.api.ui.component.EditField` class.
2. Create an instance of an `EditField`.

```
EditField edit = new EditField("EditField: ", "", 10, EditField.FILTER_DEFAULT);
```

Create a password field

1. Import the `net.rim.device.api.ui.component.PasswordEditField` class.
2. Create an instance of a `PasswordEditField`.
For example, the following instance uses a constructor that lets you provide a default initial value for the `PasswordEditField`:

```
PasswordEditField pwd = new PasswordEditField("PasswordEditField: ", "");
```

Create a text field for AutoText

If a text field supports AutoText, when users press the Space key twice, the BlackBerry® device inserts a period, capitalizes the next letter after a period, and replaces words as defined in the AutoText application.

1. Import the following classes:
 - `net.rim.device.api.ui.component.AutoTextEditField`
 - `net.rim.device.api.ui.autotext.AutoText`
 - `net.rim.device.api.ui.component.BasicEditField`
2. Create an instance of an `AutoTextEditField`.

```
AutoTextEditField autoT = new AutoTextEditField("AutoTextEditField: ", "");
```

Create a progress indicator

1. Import the `net.rim.device.api.ui.component.GaugeField` class.
2. Create an instance of a `GaugeField`.

```
GaugeField percentGauge = new GaugeField("Percent: ", 1, 100, 29, GaugeField.PERCENT);
```

Create a text label

1. Import the `net.rim.device.api.ui.component.LabelField` class.
2. Create an instance of a `LabelField` to add a text label to a screen.

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

Create a list box

Use a list box to display a list from which users can select one or more values.

1. Import the following classes:
 - `java.lang.String`
 - `net.rim.device.api.ui.component.ListField`
 - `net.rim.device.api.ui.container.MainScreen`
2. Import the `net.rim.device.api.ui.component.ListFieldCallback` interface.
3. Create a class that implements the `ListFieldCallback` interface.

```
private class ListCallback implements
ListFieldCallback {
```

4. Create the items that you want to display in a `ListField`.

```
String fieldOne = new String("Marco Cacciaccarro");
String fieldTwo = new String("Meredith Wagler");
```

5. Create an instance of a `ListField`.

```
ListField myList = new ListField(0, ListField.MULTI_SELECT);
```

6. Create an instance of a `ListCallback`.

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

7. Set the call back of the `ListField` to be the `ListCallback`.

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

8. Use the `ListCallback` object to add items to the `ListField`.

```
myCallback.add(myList, fieldOne);
myCallback.add(myList, fieldTwo);
```

9. Add the `ListField` to the `MainScreen`.

```
mainScreen.add(myList);
```

Create a field to display a tree view

Use a tree view to display objects, such as a folder structure, in a hierarchical manner. A `TreeField` contains nodes. The highest node is the root node. A node in the tree can have child nodes under it. A node that has a child is a parent node.

1. Import the following classes:
 - `net.rim.device.api.ui.component.TreeField`
 - `java.lang.String`
 - `net.rim.device.api.ui.container.MainScreen`
2. Import the `net.rim.device.api.ui.component.TreeFieldCallback` interface.
3. Implement the `TreeFieldCallback` interface.
4. Invoke `TreeField.setExpanded()` on the `TreeField` object to specify whether a folder is collapsible. Create a `TreeField` object and multiple child nodes to the `TreeField` object. We then invoke `TreeField.setExpanded()` using `node4` as a parameter to collapse the folder.

```
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);
```

- To repaint a `TreeField` when a node changes, create a class that implements the `TreeFieldCallback` interface and implement the `TreeFieldCallback.drawTreeItem` method. The `TreeFieldCallback.drawTreeItem` method uses the cookie for a tree node to draw a `String` in the location of a node. The `TreeFieldCallback.drawTreeItem` method invokes `Graphics.drawText()` to draw the `String`.

```
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);
    }
}
```

Add a UI component to a screen

- Import the following classes:
 - `net.rim.device.api.ui.component.CheckboxField`
 - `net.rim.device.api.ui.container.MainScreen`
- Create an instance of a UI component.

```
CheckboxField myCheckbox = new CheckboxField("First checkbox", true);
```

- Add the UI component to your extension of a `Screen` class.

```
mainScreen.add(myCheckbox);
```

Create a custom field

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

- Import the following classes:
 - `net.rim.device.api.ui.Field`
 - `java.lang.String`
 - `net.rim.device.api.ui.Font`
 - `java.lang.Math`
 - `net.rim.device.api.ui.Graphics`
- Import the `net.rim.device.api.ui.DrawStyle` interface.

3. Extend the `Field` class, or one of its subclasses, implementing the `DrawStyle` interface to specify the characteristics of the custom field and turn on drawing styles.

```
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;
}
```

4. Implement constructors to define a label, shape, and style of the custom button.

```
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.getAdvance(_label);
}
```

5. Implement `layout()` to specify the arrangement of field data. Perform the most complex calculations in `layout()` instead of in `paint()`. The manager of the field invokes `layout()` to determine how the field arranges its contents in the available space. Invoke `Math.min()` to return the smaller of the specified width and height, and the preferred width and height of the field. Invoke `Field.setExtent(int, int)` to set the required dimensions for the field.

```
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 );
}
```

6. Implement `getPreferredWidth()`, using the relative dimensions of the field label to make sure that the label does not exceed the dimensions of the component. Use a switch block to determine the preferred width based on the shape of the custom field. For each type of shape, use an if statement to compare dimensions and determine the preferred width for the custom field.

```
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;
    }
}
```

7. Implement `getPreferredHeight()`, using the relative dimensions of the field label to determine the preferred height. Use a switch block to determine the preferred height based on the shape of the custom field. For each type of shape, use an if statement to compare dimensions and determine the preferred height for the custom field.

```
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;
}
```

8. Implement `paint()`. The manager of a field invokes `paint()` to redraw the field when an area of the field is marked as invalid. Use a switch block to repaint a custom field based on the shape of the custom field. For a field that has a triangle or octagon shape, use the width of the field to calculate the horizontal and vertical position of a lines start point and end point. Invoke `graphics.drawLine()` and use the start and end points to draw the lines that define the custom field.

For a field that has a rectangular shape, invoke `graphics.drawRect()` and use the width and height of the field to draw the custom field. Invoke `graphics.drawText()` and use the width of the field to draw a string of text to an area of the field

```
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 );
}
```

9. Implement the `Fieldset()` and `get()` methods. Implement the `Field.getLabel()`, `Field.getShape()`, `Field.setLabel(String label)`, and `Field.setShape(int shape)` methods to return the instance variables of the custom field.

```

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;
}

```

Add a menu item to a BlackBerry Device Software application

1. Import the required classes and interfaces.

```

import net.rim.blackberry.api.menuitem.*;
import net.rim.blackberry.api.pdap.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;

```

2. Extend the abstract `ApplicationMenuItem` class, to create a menu item. Override the `ApplicationMenuItem()` constructor with an integer to specify the position of the menu item in the menu. A higher number positions the menu item lower in the menu.

```

public class SampleMenuItem extends ApplicationMenuItem
{
    SampleMenuItem()
    {
        super(20);
    }
}

```

3. Implement the `toString()` method, to specify text that the menu item displays.

```

public String toString()
{
    return "Open the Contacts Demo application";
}

```

4. Invoke `getInstance()`, to retrieve the application repository.

```

ApplicationMenuItemRepository repository =
    ApplicationMenuItemRepository.getInstance();

```

5. Create an instance of class to extend the `MenuItem` class.

```
ContactsDemoMenuItem contactsDemoMenuItem = new
    ContactsDemoMenuItem();
```

6. Invoke `ApplicationMenuItemRepository.addItem()`, to add the menu item to the relevant BlackBerry device application repository.

```
repository.addItem(ApplicationMenuItemRepository
    MENUITEM_ADDRESSCARD_VIEW, contactsDemoMenuItem);
```

7. Implement the `run()` method to specify the behavior of the menu item. In the following code sample, when a user clicks the new menu item and a `Contact` exists, the `ContactsDemo` application receives the events and invokes `ContactsDemo.enterEventDispatcher()`.

```
public Object run(Object context)
{
    BlackBerryContact c = (BlackBerryContact)context;
    if ( c != null )
    {
        new ContactsDemo().enterEventDispatcher();
    }
    else
    {
        throw new IllegalStateException( "Context is null, expected a Contact
            instance");
    }
    Dialog.alert("Viewing an email message in the email view");
    return null;
}
```

Adding a menu item to a BlackBerry Device Software application

You can add a menu item to a BlackBerry® Device Software application by using the menu item API in the `net.rim.blackberry.api.menuitem` package. For example, you can add a menu item called `View Sales Order` to the contacts application on a BlackBerry device. When a user clicks `View Sales Order`, a CRM application opens and displays a list of sales orders for that contact.

The `ApplicationMenuItemRepository` class provides constants that specify the BlackBerry Device Software application that your menu item should appear in. For example, the `MENUITEM_MESSAGE_LIST` constant specifies that the menu item should appear in the message list application.

Register a menu item

1. Import the following classes:
 - `net.rim.blackberry.api.menuitem.ApplicationMenuItemRepository`
 - `net.rim.device.api.ui.MenuItem`
2. Create a class that extends the `MenuItem` class.

```
ContactsDemoMenuItem contactsDemoMenuItem = new ContactsDemoMenuItem();
```

3. Invoke `ApplicationMenuItemRepository.getInstance` to retrieve the BlackBerry® device application repository.

```
ApplicationMenuItemRepository repository =
ApplicationMenuItemRepository.getInstance();
```

4. Create an instance of the class that extends the `MenuItem` class.

```
ContactsDemoMenuItem contactsDemoMenuItem = new ContactsDemoMenuItem();
```

5. Invoke `ApplicationMenuItemRepository.addItem()` to add the custom menu item to the BlackBerry device application repository.

```
repository.addItem(ApplicationMenuItemRepository.MENUITEM_ADDRESSCARD_VIEW,
contactsDemoMenuItem);
```

6. Invoke `ApplicationMenuItemRepository.addItem()`. Use the `MENUITEM_MAPS` field to add the menu item to the BlackBerry device application.

```
repository.addItem(ApplicationMenuItemRepository.MENUITEM_MAPS,
contactsDemoMenuItem);
```

Arrange UI components

You can arrange components on a screen using BlackBerry® API layout managers.

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

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

1. Import the following classes:

- `net.rim.device.api.ui.Manager`
- `net.rim.device.api.ui.container.HorizontalFieldManager`
- `net.rim.device.api.ui.component.ButtonField`

2. Create an instance of a `HorizontalFieldManager` and

```
HorizontalFieldManager _fieldManagerBottom = new HorizontalFieldManager();
```

3. Invoke the `add()` method to add the `HorizontalFieldManager` to a screen.

```
myScreen.add(_fieldManagerBottom);
```

4. Create an instance of a `ButtonField`.

```
ButtonField mySubmitButton = new ButtonField("Submit");
```

5. Add the `ButtonField` to the `HorizontalFieldManager`.

```
_fieldManagerBottom.add(mySubmitButton);
```

Create a layout manager

1. Import the following classes:
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.container.VerticalFieldManager`
 - `net.rim.device.api.ui.component.BitmapField`
 - `net.rim.device.api.ui.container.HorizontalFieldManager`
 - `net.rim.device.api.ui.container.FlowFieldManager`
 - `net.rim.device.api.ui.container.DialogFieldManager`
 - `net.rim.device.api.ui.container.MainScreen`
2. Create an instance of the appropriate `Manager` subclass.
3. Add UI components to the layout manager.
4. Add the layout manager to the screen. Create a `VerticalFieldManager` to organize fields from top to bottom in a single column. Add `BitmapField` instances to the `VerticalFieldManager`. Invoke `MainScreen.add()` to add the `VerticalFieldManager` to the `MainScreen` of the application.

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

Aligning a field to a line of text

You can create an application that can align a `Field` object to the natural beginning of a line of text by using the flag `Field.FIELD_LEADING`. For example, if you create a `Field` with the alignment flag `Field.FIELD_LEADING`, and add the `Field` to a `VerticalFieldManager`, if the application starts using either English or Chinese locales for example, the `Field` aligns to the left side of the screen. If the application starts using either Arabic or Hebrew locales, the `Field` aligns to the right side of the screen.

Events

Respond to navigation events

You can use the `Screen` navigation methods to create a BlackBerry® device application. If your existing BlackBerry device application implements the `TrackwheelListener` interface, update your BlackBerry device application to use the `Screen` navigation methods.

1. Import the `net.rim.device.api.ui.Screen` class.
2. Manage navigation events by extending the `net.rim.device.api.ui.Screen` class (or one of its subclasses) and overriding the following navigation methods:

```
navigationClick(int status, int time)
navigationUnclick(int status, int time)
navigationMovement(int dx, int dy, int status, int time)
```

Determine the type of input method

1. Import one or more of the following classes:
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.Field`
2. Import the `net.rim.device.api.system.KeypadListener` interface.
3. Implement the `net.rim.device.api.system.KeypadListener` interface.
4. Extend the `Screen` class, the `Field` class, or both.
5. In your implementation of one of the `navigationClick`, `navigationUnclick`, or `navigationMovement` methods, perform a bitwise AND operation on the `status` parameter to yield more information about the event. Implement the `navigationClick(int status, int time)` method to determine if the trackwheel or a four way navigation input device (for example, a trackball) triggers an event.

```
public boolean navigationClick(int status, int time) {
    if ((status & KeypadListener.STATUS_TRACKWHEEL) ==
        KeypadListener.STATUS_TRACKWHEEL) {
        //Input came from the trackwheel
    } else if ((status & KeypadListener.STATUS_FOUR_WAY) ==
```

```

KeypadListener.STATUS_FOUR_WAY) {
    //Input came from a four way navigation input device
}
return super.navigationClick(status, time);
}

```

For more information about status modifiers for the `net.rim.device.api.system.KeypadListener` class, see the API reference for the BlackBerry® Java® Development Environment.

Respond to BlackBerry device user interaction

1. Import the `net.rim.device.api.ui.Screen` class.
2. Use the `Screen` class and its subclasses to provide a menu for the BlackBerry® device users to perform actions.

Touch screen events

Types of touch screen events

A `TouchEvent` class represents a touch input action that a BlackBerry® device user performs.

A `TouchGesture` class represents an event that is a combination of touch input actions that a BlackBerry device user performs.

In the following table, all the events are in the `net.rim.device.api.ui` package.

Action	Event	Result
touch the screen lightly	<code>TouchEvent.DOWN</code>	This action highlights an item. In a text field, if users touch the screen near the cursor, an outlined box displays around the cursor. This box helps users reposition the cursor more easily.
tap the screen	<code>TouchGesture.TAP</code>	In applications that support a full-screen view, such as BlackBerry® Maps and the BlackBerry® Browser, this action hides and shows the shortcut bar.
tap the screen twice	<code>TouchGesture.TAP</code>	On a web page, map, picture, or presentation attachment, this action zooms in to the web page, map, picture, or presentation attachment.

Action	Event	Result
click (press) the screen	<code>TouchEvent.CLICK</code>	<p>This action initiates an action. For example, when users click an item in a list, the screen that is associated with the item appears. This action is equivalent to clicking the trackwheel, trackball or trackpad.</p> <p>On a map, picture, or presentation attachment, this action zooms in to the map, picture, or presentation attachment. On a web page, this action zooms in to the web page or follows a link.</p> <p>In a text field, this action positions the cursor. If the field contains text, an outlined box appears around the cursor.</p>
slide a finger up or down quickly on the screen	<code>TouchGesture.SWIPE_NORTH</code> <code>TouchGesture.SWIPE_SOUTH</code>	<p>Sliding a finger up quickly scrolls or displays the next screen. Sliding a finger down quickly scrolls or displays the previous screen.</p> <p>When the keyboard appears, sliding a finger down quickly hides the keyboard and displays the shortcut bar.</p>
slide a finger to the left or right quickly on the screen	<code>TouchGesture.SWIPE_WEST</code> <code>TouchGesture.SWIPE_EAST</code>	<p>This action scrolls, or displays the next or previous picture or message, or the next or previous day, week, or month in a calendar.</p>
hold a finger on an item	<code>TouchGesture.HOVER</code>	<p>On the shortcut bar, this action displays a tooltip that describes the action that the icon represents.</p> <p>In a message list, if users hold a finger on the sender or subject of a message, the BlackBerry device searches for the sender or subject.</p>
touch and drag an item on the screen	<code>TouchEvent.MOVE</code>	<p>This action moves the content on the screen in the corresponding direction. For example, when users touch and drag a menu item, the list of menu items moves in the same direction.</p>

Action	Event	Result
		In a text field, this action moves the outlined box and the cursor in the same direction.
touch the screen in two locations at the same time	TouchEvent.DOWN TouchEvent.MOVE	This action highlights the text or the list of items, such as messages, between the two locations. To add or remove text or items from the selection, users can touch the screen at another location.
touch the screen in two locations at the same time and drag both fingers	TouchEvent.DOWN TouchEvent.MOVE	This action moves two items on the screen in the corresponding directions.

Respond to touch screen events

Respond to system events while the user touches the screen

1. Import the following classes:
 - net.rim.device.api.ui.TouchEvent
 - net.rim.device.api.ui.Field
 - net.rim.device.api.ui.Manager
 - net.rim.device.api.ui.Screen
 - net.rim.device.api.ui.component.Dialog
2. Create a class that extends the Manager class, the Screen class, the Field class, or one of the Field subclasses.
3. In your implementation of the touchEvent(TouchEvent message) method, invoke TouchEvent.getEvent().
4. Check if the value that TouchEvent.getEvent() returns is equal to TouchEvent.CANCEL.

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.CANCEL:
            Dialog.alert("System event occurred while processing touch events");
            return true;
        }
    return false;
}
```

Respond to a user sliding a finger up quickly on the screen

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.TouchGesture`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.
3. In your implementation of the `touchEvent(TouchEvent message)` method, invoke `TouchEvent.getEvent()`.
4. Check if the value that `TouchGesture.getSwipeDirection()` returns is equal to `TouchGesture.SWIPE_NORTH`.

```
public class newButtonField extends ButtonField {

protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.GESTURE:
            TouchGesture gesture = message.getGesture();
            switch(gesture.getEvent()) {
                case TouchGesture.SWIPE:
                    if(gesture.getSwipeDirection() == TouchGesture.SWIPE_NORTH) {
                        Dialog.alert("Upward swipe occurred");
                        return true;
                    }
            }
            return false;
        }
    }
}
```

Respond to a user sliding a finger down quickly on the screen

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.TouchGesture`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.

```
public class newButtonField extends ButtonField {
```

3. In your implementation of the `touchEvent(TouchEvent message)` method, invoke `TouchEvent.getEvent()`.
4. Check if the value that `TouchGesture.getSwipeDirection()` returns is equal to `TouchGesture.SWIPE_SOUTH`.

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.GESTURE:
            TouchGesture gesture = message.getGesture();
            switch(gesture.getEvent()) {
                case TouchGesture.SWIPE:
                    if(gesture.getSwipeDirection() == TouchGesture.SWIPE_SOUTH) {
                        Dialog.alert("Downward swipe occurred");
                        return true;
                    }
            }
            return false;
    }
}
```

Respond to a user sliding a finger to the left quickly on the screen

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.TouchGesture`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.

```
public class newButtonField extends ButtonField {
```

3. In your implementation of the `touchEvent(TouchEvent message)` method, invoke `TouchEvent.getEvent()`.
4. Check if the value that `TouchGesture.getSwipeDirection()` returns is equal to `TouchGesture.SWIPE_EAST`.

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.GESTURE:
            TouchGesture gesture = message.getGesture();
            switch(gesture.getEvent()) {
                case TouchGesture.SWIPE:
                    if(gesture.getSwipeDirection() == TouchGesture.SWIPE_EAST) {
                        Dialog.alert("Eastward swipe occurred");
                        return true;
                    }
            }
    }
}
```

```

        return false;
    }
}

```

Respond to a user sliding a finger to the right quickly on the screen

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.TouchGesture`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.

```
public class newButtonField extends ButtonField {
```

3. In your implementation of the `touchEvent(TouchEvent message)` method, invoke `TouchEvent.getEvent()`.
4. Check if the value that `TouchGesture.getSwipeDirection()` returns is equal to `TouchGesture.SWIPE_WEST`.

```

protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.GESTURE:
            TouchGesture gesture = message.getGesture();
            switch(gesture.getEvent()) {
                case TouchGesture.SWIPE:
                    if(gesture.getSwipeDirection() == TouchGesture.SWIPE_WEST) {
                        Dialog.alert("Westward swipe occurred");
                        return true;
                    }
            }
            return false;
    }
}
}

```

Respond to a user clicking the screen

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`

2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.

```
public class newButtonField extends ButtonField {
```

3. In your implementation of the `touchEvent(TouchEvent message)` method, invoke `TouchEvent.getEvent()`.
4. Check if the value that `TouchEvent.getEvent()` returns is equal to `TouchEvent.CLICK`.

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.CLICK:
            Dialog.alert("CLICK occurred");
            return true;
    }
    return false;
}
```

Respond to a user touching the screen twice quickly

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.TouchGesture`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.

```
public class newButtonField extends ButtonField {
```

3. In your implementation of the `touchEvent(TouchEvent message)` method, check for the occurrence of a `TouchGesture.TAP` event and that `TouchGesture.getTapCount` returns 2.

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.GESTURE:
            TouchGesture gesture = message.getGesture();
            switch(message.getEvent()) {
                case TouchGesture.TAP:
                    if(gesture.getTapCount() == 2) {
                        Dialog.alert("Double tap occurred");
                        return true;
                    }
            }
    }
    return false;
}
```

Respond to a user touching and dragging an item on the screen

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.
3. In your implementation of the `TouchEvent(TouchEvent message)` method, invoke `TouchEvent.getEvent()`.
4. Check if the value that `TouchEvent.getEvent()` returns is equal to `TouchEvent.MOVE`.

```
public class newButtonField extends ButtonField {
```

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.MOVE:
            Dialog.alert("Move event occurred");
            return true;
    }
    return false;
}
```

Respond to a user touching the screen lightly

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.TouchGesture`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.
3. In your implementation of the `TouchEvent(TouchEvent message)` method, invoke `TouchEvent.getEvent()`.
4. Check if the value that `TouchEvent.getEvent()` returns is equal to `TouchEvent.DOWN`.

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.DOWN:
            Dialog.alert("Touch occurred");
    }
}
```

```

        return true;
    }
    return false;
}

```

Respond to a scroll action

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.TouchGesture`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
2. Create a class that extends the `Manager` class.

```
public class newManager extends Manager {
```

3. In your implementation of the `touchEvent(TouchEvent message)` method, check if the value `TouchEvent.getEvent()` returns is equal to `TouchEvent.MOVE`.

```

protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
        case TouchEvent.MOVE:
            return true;
    }
    return false;
}

```

Respond to a user touching the screen in two locations at the same time

1. Import the following classes:
 - `net.rim.device.api.ui.TouchEvent`
 - `net.rim.device.api.ui.Field`
 - `net.rim.device.api.ui.Manager`
 - `net.rim.device.api.ui.Screen`
 - `net.rim.device.api.ui.component.Dialog`
2. Create a class that extends the `Manager` class, the `Screen` class, the `Field` class, or one of the `Field` subclasses.


```
public class newButtonField extends ButtonField {
```
3. In your implementation of the `touchEvent(TouchEvent message)` method, check if the following method invocations return values greater than zero: `TouchEvent.getX(1)`, `TouchEvent.getY(1)`, `TouchEvent.getX(2)`, `TouchEvent.getY(2)`.

```
protected boolean touchEvent(TouchEvent message) {
    switch(message.getEvent()) {
    case TouchEvent.MOVE:
        if(message.getX(1) > 0 && message.getY(1) > 0) {
            Dialog.alert("First finger touched/moved");
        }
        if(message.getX(2) > 0 && message.getY(2) > 0) {
            Dialog.alert("Second finger touched/moved");
        }
        return true;
    }
    return false;
}
```

Keyboard on a BlackBerry device with a touch screen

A BlackBerry device application can have more than one Screen. Each Screen has one touch screen keyboard.

Change the state of the keyboard on a BlackBerry device with a touch screen

1. Import the following classes:
 - `net.rim.device.api.ui.VirtualKeyboard`
 - a screen class, such as `net.rim.device.api.ui.container.MainScreen`
2. Create an instance of `VirtualKeyboard`.

```
VirtualKeyboard virtKbd;
```

3. Store an instance of a Screen class.

```
MainScreen _screen = new MainScreen( MainScreen.VERTICAL_SCROLL );
```

4. Invoke `getVirtualKeyboard()` and store the return value in the `VirtualKeyboard` variable.

```
virtKbd = _screen.getVirtualKeyboard();
```

5. Perform one of the following tasks:

Task	Steps
Retrieve the state of the touch screen keyboard.	Invoke <code>VirtualKeyboard.getVisibility()</code> .
Change the state of the touch screen keyboard.	Invoke <code>VirtualKeyboard.setVisibility(int)</code> .
Inherit the state of the touch screen keyboard.	Invoke <code>VirtualKeyboard.setVisibility(VirtualKeyboard.IGNORE)</code> .

Display the keyboard on a BlackBerry device with a touch screen

1. Import the following classes:
 - `net.rim.device.api.ui.VirtualKeyboard`
 - a `Screen` class, such as `net.rim.device.api.ui.container.MainScreen`
2. Create an instance of `VirtualKeyboard`.

```
VirtualKeyboard virtKbd;
```

3. Store an instance of a screen class.

```
MainScreen _screen = new MainScreen( MainScreen.VERTICAL_SCROLL );
```

4. Invoke `getVirtualKeyboard()` and store the return value in the `VirtualKeyboard` variable.

```
virtKbd = _screen.getVirtualKeyboard();
```

5. Perform one of the following tasks:

Task	Steps
Always display the touch screen keyboard.	Invoke <code>VirtualKeyboard.setVisibility(VirtualKeyboard.SHOW_FORCE)</code> .
Display the touch screen keyboard on a new screen.	Invoke <code>VirtualKeyboard.setVisibility(int)</code> using one of the following constants as a parameter: <code>VirtualKeyboard.SHOW</code> or <code>VirtualKeyboard.SHOW_FORCE</code> .

Hide the keyboard on a BlackBerry device with a touch screen

1. Import the following classes:
 - `net.rim.device.api.ui.VirtualKeyboard`
 - a screen class, such as `net.rim.device.api.ui.container.MainScreen`
2. Create an instance of `VirtualKeyboard`.

```
VirtualKeyboard virtKbd;
```

3. Store an instance of a screen class.

```
MainScreen _screen = new MainScreen( MainScreen.VERTICAL_SCROLL );
```

4. Invoke `getVirtualKeyboard()` and store the return value in the `VirtualKeyboard` variable.

```
virtKbd = _screen.getVirtualKeyboard();
```

5. Perform one of the following tasks:

Task	Steps
Always hide the touch screen keyboard.	Invoke <code>VirtualKeyboard.setVisibility(VirtualKeyboard.HIDE_FORCE)</code> .
Hide the touch screen keyboard on a new screen.	Invoke <code>VirtualKeyboard.setVisibility(int)</code> using one of the following constants as a parameter: <code>VirtualKeyboard.HIDE</code> or <code>VirtualKeyboard.HIDE_FORCE</code> .

Spell check

You can use the items of the `net.rim.blackberry.api.spellcheck` package to add spell check functionality to an application. The `SpellCheckEngine` interface lets an application check the spelling of UI field values and provides a BlackBerry® device user with options for spelling corrections. The `SpellCheckUI` interface lets an application provide a UI that allows a BlackBerry device user to resolve spell check issues by interacting with the `SpellCheckEngine` implementation.

For more information about using the Spell Check API, see the `SpellCheckDemo` sample application, included with the BlackBerry® Java® Development Environment version 4.3.1 or later.

Add spell check functionality

1. Import the following classes:
 - `net.rim.blackberry.api.spellcheck.SpellCheckEngineFactory`
 - `java.lang.StringBuffer`
2. Import the following interfaces:
 - `net.rim.blackberry.api.spellcheck.SpellCheckEngine`
 - `net.rim.blackberry.api.spellcheck.SpellCheckUI`
 - `net.rim.blackberry.api.spellcheck.SpellCheckUIListener`
3. Create variables for spell check objects.

```
SpellCheckEngine _spellCheckEngine;
SpellCheckUI _spellCheckUI;
```

4. Invoke `createSpellCheckUI()`.

```
_spellCheckUI = SpellCheckEngineFactory.createSpellCheckUI();
```

5. To notify an application when a spell check event occurs, invoke `addSpellCheckUIListener()` with a `SpellCheckUIListener` object as a parameter.

```
_spellCheckUI.addSpellCheckUIListener(new SpellCheckUIListener());
```

- To let an application spell check UI fields and suggest spelling corrections to a BlackBerry device user, obtain a `SpellCheckEngine` object, invoke `getSpellCheckEngine()`.

```
_spellCheckEngine = _spellCheckUI.getSpellCheckEngine();
```

- To use a correction for a misspelled word, invoke `SpellCheckEngine.learnCorrection()`. Use the parameters `newStringBuffer(text)`, `newStringBuffer(correction)`, where `text` represents the misspelled word, and `correction` represents the correct word.

```
_spellCheckEngine.learnCorrection(new StringBuffer(text), new StringBuffer(correction));
```

- To perform spell check operations on a field, invoke `SpellCheckUI.spellCheck()`, with a `field` as a parameter.

```
_spellCheckUI.spellCheck(field);
```

- To accept a misspelled word as correctly spelled, invoke `SpellCheckEngine.learnWord()`, with the word to learn as a parameter.

```
_spellCheckEngine.learnWord(new StringBuffer(word));
```

Listen for a spell check event

- Import the following classes:
 - `java.lang.StringBuffer`
 - `net.rim.device.api.ui.UiApplication`
 - `net.rim.device.api.ui.Field`
- Import the following interfaces:
 - `net.rim.blackberry.api.spellcheck.SpellCheckUIListener`
 - `net.rim.blackberry.api.spellcheck.SpellCheckEngine`
- Create a method that returns the `SpellCheckUIListener.LEARNING_ACCEPT` constant when the `SpellCheckEngine` learns a new word.

```
public int wordLearned(SpellCheckUI ui, StringBuffer word) {
    UiApplication.getUiApplication().invokeLater(new popUpRunner("Word learned"));
    return SpellCheckUIListener.LEARNING_ACCEPT;
}
```

- Create a method that returns the `SpellCheckUIListener.LEARNING_ACCEPT` constant when the `SpellCheckEngine` learns a correction for a word.

```
public int wordCorrectionLearned(SpellCheckUI ui, StringBuffer word, StringBuffer correction) {
    UiApplication.getUiApplication().invokeLater(new popUpRunner("Correction learned"));
    return SpellCheckUIListener.LEARNING_ACCEPT;
}
```

5. Create a method that returns the `SpellCheckUListener.ACTION_OPEN_UI` constant when the `SpellCheckEngine` finds a misspelled word.

```
public int misspelledWordFound(SpellCheckUI ui, Field field, int offset, int len){
    UiApplication.getUiApplication().invokeLater(new popUpRunner("Misspelled word
found"));
    return SpellCheckUListener.ACTION_OPEN_UI;
}
```

Accessibility

Integrating with assistive technology software

On a BlackBerry® device with BlackBerry® Device Software version 4.6.1 or later, a BlackBerry device application that uses the UI component classes can provide information to an assistive technology application when a UI component changes. One example of an assistive technology application is a screen reader. The UI component classes include the `ObjectChoiceField`, `RadioButtonField`, and `RichTextField` classes in the `net.rim.device.api.ui.component` package.

If a BlackBerry device application uses the UI component classes and does not extend the classes, you do not need to alter the application to integrate with an assistive technology application. For example, if a BlackBerry device application uses the `net.rim.device.api.ui.component.TextField` class and does not extend the class, an assistive technology application can access information when the `TextField` object changes.

If a BlackBerry device application extends any of the UI component classes, and thus uses custom UI components, you must use the `net.rim.device.api.ui.accessibility` package to provide an assistive technology application with access to information about a custom UI component that changes. For example, a BlackBerry device application that uses a `myTextField` class that extends `TextField` must use the `net.rim.device.api.ui.accessibility` package to provide access to data when text in the `myTextField` object changes.

You must first determine whether the BlackBerry device supports the Accessibility APIs. Create an instance of the `net.rim.device.api.ui.accessibility.AccessibilityManager` class and invoke `AccessibilityManager.isAccessibilitySupported()`. If the method returns `true`, the BlackBerry device application can make data available to an assistive technology application.

To specify the type of information the application can provide to an assistive technology application, you must implement the `AccessibleContext` interface. The `AccessibleContext` interface enables a BlackBerry device application to provide information such as the name of the custom UI component, the parent component of the custom UI component, and the number of child components of a custom UI component. For example, invoking the method that implements the `AccessibleContext.getAccessibleChildCount()` method should return 4 if a `Screen` for a BlackBerry device application contains four `myTextField` objects.

You must implement the `AccessibleContext` interface to provide an assistive technology application with access to the textual, tabular, or numerical data from a custom UI component that changes. You must also implement the `AccessibleText`, `AccessibleTable`, and `AccessibleValue` interfaces.

After you implement the interfaces, you can invoke the methods that implement the `AccessibleContext.getAccessibleText()`, `AccessibleContext.getAccessibleTable()`, and `AccessibleContext.getAccessibleValue()` methods to return an object that provides access to the textual, tabular, and numerical data. For example, a BlackBerry device application that extends `TextField` can use an implementation of the `AccessibleText.getAtIndex()` and `AccessibleText.getCaretPosition()` methods to retrieve the `String` at a specific location in text and the position of the cursor in text.

To send notifications to an assistive technology application when a custom UI component changes, you must implement the `net.rim.device.api.ui.accessibility.AccessibleEventListener` interface. Various events can cause a BlackBerry device application to send notifications, including a change to the text in a field, a change in the location of the cursor in the text in a field, and a change to a field that a UI component contains. A notification about a change can contain different kinds of information, including the name of the custom UI component that changes, the type of change, the value of the data before the change, and the value of the data after the change.

Code sample: Making information about text changes available to an assistive technology application

```
class makeAccessibleText implements AccessibleText {
    ...
    public String getAtIndex(int part, int index){
        switch (part){
            case AccessibleText.CHAR:
                return _text.charAt(index);

            case AccessibleText.WORD:
                String[] words = splitWords(_text);
                return words[index];

            case AccessibleText.LINE:
                String[] lines = splitLines(_text);
                return lines[index];
        }
        return null;
    }
}

class makeAccessible implements AccessibleContext {
    ...
    makeAccessibleText makeAccessibleTextObj = new makeAccessibleText();
    public AccessibleText getAccessibleText(){
        return makeAccessibleTextObj;
    }
    ...
}
```

Notifying an assistive technology application when the UI changes

You can enable a BlackBerry® device application that uses custom UI components to send information to an assistive technology application by using the classes and interfaces in the `net.rim.device.api.ui.accessibility` package. When a custom UI component changes, an assistive technology application receives a notification about the change and can retrieve more information about the change from the custom UI component.

The notification contains the following information:

- name of the custom UI component
- type of event, for example, a change in the cursor position, or a change in the name of the custom UI component
- value of a custom UI component before the event
- value of a custom UI component after the event

For example, if a BlackBerry device application uses a custom class called `myTextField` that extends the `TextField` class, when a BlackBerry device user changes the text in a `myTextField` instance, an assistive technology application receives notification of the change and can retrieve information such as the text that the user selects or changes.

UI changes that trigger a notification to an assistive technology application

The following changes to a UI component can trigger a notification to an assistive technology application:

- a change to the position of a cursor
- a change to the name
- a change to the text
- a change in a child component
- a change in the state
- a change to the numeric value

UI component states and properties

A UI component can have one or more of the following states:

- focused
- focusable
- expanded
- expandable
- collapsed
- selected
- selectable
- pushed
- checked

- editable
- active
- busy

A UI component can have one or more of the following properties:

- modal
- horizontal
- vertical
- single-line
- multi-line

Provide an assistive technology application with information about a UI change

1. Import the required interface.

```
import net.rim.device.api.ui.accessibility.AccessibleContext;
```

2. Implement `AccessibleContext` to represent the information about changes to a custom UI component that an application can make available.
3. Create variables to store the accessibility information for a custom UI component, such as state information.

```
private int _state = AccessibleState.UNSET;  
private String _accessibleName;  
private String _title;
```

4. Create the methods that add and remove the states for a custom UI component.

```
protected void addAccessibleStates(int states)  
{  
    _state = _state & ~AccessibleState.UNSET;  
    _state = _state | states;  
}  
  
protected void removeAccessibleStates(int states)  
{  
    _state = _state & ~states;  
}  
  
public void setAccessibleName(String accessibleName)  
{  
    _accessibleName = accessibleName;  
}
```

5. Implement the `get()` methods of `AccessibleContext`, to allow access to tabular, textual, or numerical information for a custom UI component. Return `null` if the method does not apply to a custom UI control. For example, if the control does not provide textual information, return `null` in `getAccessibleText()`.

```
public AccessibleTable getAccessibleTable()
{
    return null;
}

public AccessibleText getAccessibleText()
{
    return null;
}

public AccessibleValue getAccessibleValue()
{
    return null;
}
```

6. Create a method that returns an instance of the class that implements `AccessibleContext`, to provide information about a change to a custom UI component.

```
public AccessibleContext getAccessibleContext()
{
    return this;
}
```

7. Implement `AccessibleContext.getAccessibleRole()`, to provide information about the type of custom UI control.

```
public int getAccessibleRole()
{
    return AccessibleRole.PANEL;
}
```

8. Implement `AccessibleContext.getAccessibleChildCount()`, to provide information about the number of accessible child components that a custom UI component contains. It is up to you as the developer of a custom UI component to decide whether or not it contains accessible children. One criterion you can use to decide whether a child element is an accessible child component is whether a child element can gain focus and whether the user can interact with the child element independently.

```
public int getAccessibleChildCount()
{
    return _icons.size();
}
```

9. Implement `AccessibleContext.getAccessibleChildAt(int)`, to provide information about an accessible child component that a custom UI component contains.

```
public AccessibleContext getAccessibleChildAt(int arg0)
{
    return (Icon) _icons.elementAt( index );
}
```

10. Implement `AccessibleContext.getAccessibleName()`, to provide the name of a custom UI component that changes.

```
public String getAccessibleName()
{
    return _accessibleName;
}
```

11. Create a method that invokes `AccessibleEventDispatcher.dispatchAccessibleEvent(stateChange, oldState, newState, this)`, to send a notification when the state of a custom UI component changes. Use the following information as arguments to the method:

- the event
- the old state of the custom UI component
- the new state of the custom UI component
- an instance of the custom UI component

```
protected void onFocus(int direction)
{
    super.onFocus(direction);

    int oldState = _state;
    _state = _state | AccessibleState.FOCUSED;

    if(isVisible())
    {
        AccessibleEventDispatcher.dispatchAccessibleEvent(
            AccessibleContext.ACCESSIBLE_STATE_CHANGED,
            new Integer(oldState),
            new Integer(_state),
            this);
    }
}
```

Provide an assistive technology application with information about text changes

You can provide accessibility information about the custom UI controls that display text.

1. Import the required interfaces.

```
import net.rim.device.api.ui.accessibility.AccessibleContext;
import net.rim.device.api.ui.accessibility.AccessibleText;
```

`AccessibleContext` provides basic accessibility information about a custom UI control. `AccessibleText` provides information about the text in a custom UI control that displays text.

2. Create a class that extends `Field` and implements the `AccessibleContext` and `AccessibleText` interfaces.

```
public class AccessibleTextClass extends Field
    implements AccessibleContext, AccessibleText
{
}
```

3. Create variables to store the custom UI control's accessibility state and the contents.

```
private int _state;
private int _cursor, _anchor;
private String _text;
private Vector _lines;
private Vector _words;
```

4. Implement `AccessibleText.getAtIndex(int part, int index)`, to provide access to the text at a given position in the custom UI control.

```
public String getAtIndex(int part, int index)
{
    switch(part)
    {
        case AccessibleText.CHAR:
            return String.valueOf(_text.charAt(index));

        case AccessibleText.LINE:
            return (String) _lines.elementAt(index);

        case AccessibleText.WORD:
            return (String) _words.elementAt(index);
    }

    return null;
}
```

5. Implement `AccessibleText.getSelectionStart()`, to provide access to the position of the first character of text that a BlackBerry® device user selects. Return 0 if the device user cannot select text in the custom UI control.

```
public int getSelectionStart()
{
    return _anchor;
}
```

6. Implement `AccessibleText.getSelectionEnd()`, to provide access to the position of the last character of text that a BlackBerry device user selects. Return 0 if the device user cannot select text in the custom UI control.

```
public int getSelectionEnd()
{
    return _cursor;
}
```

7. Implement `AccessibleText.getCaretPosition()`, to provide access to the position of the cursor within the text. Return 0 if the device user cannot select text in the custom UI control.

```
public int getCaretPosition()
{
    return _cursor;
}
```

8. Implement `AccessibleText.getSelectionText()`, to provide access to the selected part of the text within the custom UI control.

```
public String getSelectionText()
{
    int start = getSelectionStart();
    int end = getSelectionEnd();
    if (start < end) {
        return _text.getText(start, end);
    } else {
        return _text.getText(end, start);
    }
}
```

9. Implement `AccessibleText.getLineCount()`, to provide access to the number of lines of text within the custom UI control.

```
public int getLineCount()
{
    return _lines.size();
}
```

10. Implement `AccessibleText.getCharCount()`, to provide access to the number of characters in the text within the custom UI control.

```
public int getCharCount()
{
    return _text.length();
}
```

Provide an assistive technology application with access to information from a table

You can provide accessibility information about the custom UI controls that display tabular information.

1. Import the required interfaces.

```
import net.rim.device.api.ui.accessibility.AccessibleContext;
import net.rim.device.api.ui.accessibility.AccessibleTable;
```

`AccessibleContext` provides basic accessibility information about a custom UI control. `AccessibleTable` provides information about the tabular information in a custom UI control.

2. Create a class that implements the `AccessibleContext` interface and the `AccessibleTable` interface.

```
public class AccessibleTableClass
    implements AccessibleContext, AccessibleTable
{
}
```

3. Create variables to store the accessibility information about the table.

```
private int _columnCount;
private int _rowCount;
private String[] _columnNames;
private String[][] _cells;
```

4. Implement the `getAccessible()` methods to provide the accessibility information about the table.

```
public int getAccessibleColumnCount()
{
    return _columnCount;
}

public int getAccessibleRowCount()
{
    return _rowCount;
}

public AccessibleContext[] getAccessibleColumnHeader()
{
    AccessibleContext[] result = new AccessibleContext[_columnNames.length];
    for( int i = 0; i < result.length; i++ )
    {
        result[i] = new AccessibleLabel(_columnNames[i], false);
    }
    return result;
}

public AccessibleContext[] getAccessibleRowHeader()
{
    return null;
}
```

Provide an assistive technology application with access to numeric values

You can provide accessibility information about the custom UI controls, such as progress indicators, that display numeric values.

1. Import the required interfaces.

```
import net.rim.device.api.ui.accessibility.AccessibleContext;
import net.rim.device.api.ui.accessibility.AccessibleValue;
```

`AccessibleContext` provides basic accessibility information about a custom UI control. `AccessibleValue` provides information about the numeric values in a custom UI control that displays numeric values.

2. Create a class that implements the `AccessibleContext` interface and the `AccessibleValue` interface.

```
public class GaugeFieldAccessibleValue extends Field
    implements AccessibleContext, AccessibleValue
{
}
```

3. Create variables to store the custom UI control's accessibility state and the contents.

```
private int _state;
private int _min;
private int _current;
private int _max;
```

4. Implement the `AccessibleValue.get()` methods to provide the accessibility information about the numeric values.

```
public int getCurrentAccessibleValue()
{
    return _current;
}

public int getMaxAccessibleValue()
{
    return _max;
}

public int getMinAccessibleValue()
{
    return _min;
}
```

Enable an assistive technology application to receive notification of UI events

If you are building an assistive technology application, such as a screen reader, implement the `AccessibleEventListener` interface.

1. Import the required interface.

```
import net.rim.device.api.ui.accessibility.AccessibleEventListener;
```

2. Create a class that implements the `AccessibleEventListener` interface.

```
public class ScreenReader implements AccessibleEventListener
{
}
```

3. Implement `AccessibleEventListener.accessibleEventOccurred(int event, Object oldValue, Object newValue, AccessibleContext context)`, to respond to UI events in an accessible application that registers this assistive technology application as an `AccessibleEventListener`.

```
public synchronized void accessibleEventOccurred(  
    int event,  
    Object oldValue,  
    Object newValue,  
    AccessibleContext context)  
{  
    if(context == null)  
    {  
        return;  
    }  
  
    int oldState = (oldValue instanceof Integer) ? ((Integer) oldValue).intValue()  
        : 0;  
    int newState = (newValue instanceof Integer) ? ((Integer) newValue).intValue()  
        : 0;  
  
    switch(context.getAccessibleRole())  
    {  
        case AccessibleRole.APP_ICON:  
            ScreenReaderHandler.handleAppIcon(event, oldState, newState, context);  
            break;  
  
        case AccessibleRole.ICON:  
            ScreenReaderHandler.handleIcon(event, oldState, newState, context);  
            break;  
  
        case AccessibleRole.CHECKBOX:  
            ScreenReaderHandler.handleCheckBox(event, oldState, newState, context);  
            break;  
  
        case AccessibleRole.CHOICE:  
            ScreenReaderHandler.handleChoice(event, oldState, newState, context);  
            break;  
  
        ...  
    }  
}
```

Storing data

2

Data management

You can store data to persistent memory on the BlackBerry® device. The BlackBerry Persistent Store APIs and the MIDP RMS APIs (support for JSR 37 and JSR 118) are available on all Java® based BlackBerry devices. A BlackBerry device that runs BlackBerry® Device Software version 4.2 or later provides a traditional file system and support for saving content directly to the file system via JSR 75 APIs. With either the BlackBerry Persistent Store APIs or the MIDP RMS APIs, you can store data persistently to flash memory. The data persists even if you remove the battery from the BlackBerry device.

Support for APIs to store data to persistent memory

Data management

The BlackBerry® device provides APIs for storing data to persistent memory on the BlackBerry device. The BlackBerry Persistent Store APIs and the MIDP RMS APIs (support for JSR 37 and JSR 118) are available on all Java® based BlackBerry devices. A BlackBerry device that runs BlackBerry® Device Software version 4.2 or later provides a traditional file system and support for saving content directly to the file system using JSR 75 APIs. With either the BlackBerry Persistent Store APIs or the MIDP RMS APIs, you can store data persistently to flash memory. The data persists even if you remove the battery from the BlackBerry device.

Access to memory

The BlackBerry® Java® environment is designed to inhibit applications from causing problems accidentally or maliciously in other applications or on the BlackBerry device. BlackBerry device applications can write only to the BlackBerry device memory that the BlackBerry Java Virtual Machine uses; they cannot access the virtual memory or the persistent storage of other applications (unless they are specifically granted access to do so). Custom applications can only access persistent storage or user data, or communicate with other applications, through specific APIs. Research In Motion must digitally sign applications that use certain BlackBerry APIs, to provide an audit trail of applications that use sensitive APIs.

Persistent data storage

Create a persistent data store

Each `PersistentObject` has a unique long key.

1. Import the following classes:
 - `net.rim.device.api.system.PersistentObject`
 - `net.rim.device.api.system.PersistentStore`
 - `java.lang.String`
 - `net.rim.device.api.ui.component.Dialog`
2. To create a unique long key, in the BlackBerry® Integrated Development Environment, type a string value.

```
com.rim.samples.docs.userinfo
```

3. Right-click the string and click **Convert 'com.rim.samples.docs.userinfo' to long**.
4. Include a comment in your code to indicate the string that you used to generate the unique long key.
5. To create a persistent data store, create a single static `PersistentObject` and invoke `PersistentStore.getPersistentObject`, using the unique long key as a parameter.

```
static PersistentObject store;  
static {  
    store = PersistentStore.getPersistentObject( 0xa1a569278238dad2L );  
}
```

Store persistent data

1. Import the following classes:
 - `net.rim.device.api.system.PersistentStore`
 - `net.rim.device.api.system.PersistentObject`
2. Invoke `setContents()` on a `PersistentObject`. This method replaces existing content with the new content.
3. To save the new content to the persistent store, invoke `commit()`.

```
String[] userinfo = {username, password};  
synchronized(store) {  
    store.setContents(userinfo);  
    store.commit();  
}
```

4. To use a batch transaction to commit objects to the persistent store, invoke `PersistentStore.getSynchObject()`. This method retrieves the persistent store monitor that locks the object.

- a. Synchronize on the object.
 - b. Invoke `commit()` as necessary. If any commit in the batch fails, the entire batch transaction fails.
5. To commit a monitor object separately from a batch transaction, invoke `forceCommit()` while synchronizing the monitor object.

Retrieve persistent data

1. Import the following classes:
 - `net.rim.device.api.system.PersistentObject`
 - `net.rim.device.api.ui.component.Dialog`
2. Invoke `getContents()` on a `PersistentObject`.
3. To convert to your desired format, perform an explicit cast on the object that `PersistentObject.getContents()` returns.

```
synchronized(store) {
String[] currentinfo = (String[])store.getContents();
if(currentinfo == null) {
Dialog.alert(_resources.getString(APP_ERROR));
}
else {
currentusernamefield.setText(currentinfo[0]);
currentpasswordfield.setText(currentinfo[1]);
}
}
```

Remove persistent data

If you delete the `.cod` file that defines a `PersistentStore`, then all persistent objects that the `.cod` file created are deleted.

1. Import the following classes:
 - `net.rim.device.api.system.PersistentStore`
 - `net.rim.device.api.system.PersistentObject`
2. To remove all persistent data from a BlackBerry® device application, invoke `PersistentStore.destroyPersistentObject()`, providing as a parameter a unique key for the `PersistentObject`.
3. To remove individual data, treat the data as normal objects, and remove references to it. A garbage collection operation removes the data.

MIDP record storage

Create an MIDP record store

1. Import the `javax.microedition.rms.RecordStore` class.
2. Invoke `openRecordStore()`, and specify `true` to indicate that the method should create the record store if the record store does not exist.

```
RecordStore store = RecordStore.openRecordStore("Contacts", true);
```

Add a record to a record store

1. Import the `javax.microedition.rms.RecordStore` class.
2. Invoke `addRecord()`.

```
int id = store.addRecord(_data.getBytes(), 0, _data.length());
```

Retrieve a record from a record store

1. Import the following classes:
 - `javax.microedition.rms.RecordStore`
 - `java.lang.String`
2. Invoke `getRecord(int, byte[], int)`. Pass the following parameters:
 - a record ID
 - a byte array
 - an offset

```
byte[] data = new byte[store.getRecordSize(id)];  
store.getRecord(id, data, 0);  
String dataString = new String(data);
```

Retrieve all records from a record store

1. Import the `javax.microedition.rms.RecordStore` class.
2. Import the following interfaces:
 - `javax.microedition.rms.RecordEnumeration`
 - `javax.microedition.rms.RecordFilter`
 - `javax.microedition.rms.RecordComparator`
3. Invoke `openRecordStore()`.

- Invoke `enumerateRecords()`. Pass the following parameters:
 - `filter`: specifies a `RecordFilter` object to retrieve a subset of record store records (if null, the method returns all records)
 - `comparator`: specifies a `RecordComparator` object to determine the order in which the method returns the records (if null, the method returns the records in any order)
 - `keepUpdated`: determines if the method keeps the enumeration current with the changes to the record store

```
RecordStore store = RecordStore.openRecordStore("Contacts", false);
RecordEnumeration e = store.enumerateRecords(null, null, false);
```

Collections

Retrieve a collection from persistent storage

- Import the following classes:
 - `net.rim.device.api.system.PersistentStore`
 - `java.util.Vector`
- Import the `net.rim.device.api.synchronization.SyncCollection` interface.
- To provide the BlackBerry® device application with access to the newest `SyncCollection` data from the `PersistentStore`, invoke the `PersistentStore.getPersistentObject()` method using the ID of the `SyncCollection`.

```
private PersistentObject _persist;
private Vector _contacts;
private static final long PERSISTENT_KEY = 0x266babf899b20b56L;
_persist = PersistentStore.getPersistentObject( PERSISTENT_KEY );
```

- Store the returned data in a vector object.


```
_contacts = (Vector)_persist.getContents();
```
- Create a method to provide the BlackBerry device application with the newest `SyncCollection` data before a wireless data backup session begins.

```
public void beginTransaction()
{
    _persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);
    _contacts = (Vector)_persist.getContents();
}
```

- Create code to manage the case where the `SyncCollection` you retrieve from the `PersistentStore` is empty.

```
if( _contacts == null )
{
    _contacts = new Vector();
}
```

```
_persist.setContents( _contacts );
_persist.commit();
}
```

Create a collection listener to notify the system when a collection changes

The system invokes `CollectionEventSource.addCollectionListener()` to create a `CollectionListener` for each `SyncCollection` the BlackBerry® device application makes available for wireless backup.

1. Import the `net.rim.device.api.util.ListenerUtilities` class.
2. Import the following interfaces:
 - `java.util.Vector`
 - `net.rim.device.api.collection.CollectionEventSource`
 - `net.rim.device.api.collection.CollectionListener`
 - `net.rim.device.api.synchronization.SyncCollection`
3. Create a private vector object to store the collection of `SyncCollection` listeners for the BlackBerry device application.

```
private Vector _listeners;
_listeners = new CloneableVector();
```

4. Implement the `CollectionEventSource.addCollectionListener()` method, making sure the method adds a `CollectionListener` to the `Vector` that contains listeners. In the following code sample, we implement `CollectionEventSource.addCollectionListener()` to invoke `ListenerUtilities.fastAddListener()` to add a listener to the `Vector` that contains listeners.

```
public void addCollectionListener(Object listener)
{
    _listeners = ListenerUtilities.fastAddListener( _listeners, listener );
}
```

Remove a collection listener that notifies the system when a collection changes

When a `CollectionListener` is no longer required, the system invokes `CollectionEventSource.removeCollectionListener()`.

1. Import the following classes:
 - `net.rim.device.api.util.ListenerUtilities`
 - `java.util.Vector`
2. Import the following interfaces:
 - `net.rim.device.api.collection.CollectionEventSource`
 - `net.rim.device.api.collection.CollectionListener`
3. Implement the following interfaces:
 - `net.rim.device.api.collection.CollectionEventSource`
 - `net.rim.device.api.collection.CollectionListener`

4. Implement the `CollectionEventSource.removeCollectionListener()` method, using the `ListenerUtilities.removeListener()` method to remove a `CollectionListener` from the `Vector` that contains `SyncCollection` listeners for the BlackBerry® device application. In the following code sample, we implement `CollectionEventSource.removeCollectionListener()` to invoke `ListenerUtilities.removeListener()` to remove a listener from the `Vector` that contains listeners

```
public void removeCollectionListener(Object listener)
{
    _listeners = ListenerUtilities.removeListener( _listeners, listener );
}
```

Notify the system when a collection changes

1. Import the `net.rim.device.api.collection.CollectionListener` interface.
2. To notify the system when an element is added to a `SyncCollection`, invoke `CollectionListener.elementAdded()`.

```
for( int i=0; i<_listeners.size(); i++ )
{
    CollectionListener cl = (CollectionListener)_listeners.elementAt( i );
    cl.elementAdded( this, object );
}
return true;
}
```

3. To notify the system when an element in a `SyncCollection` is replaced, invoke `CollectionListener.elementUpdated()`.
4. Invoke `CollectionListener.elementRemoved()`.

Runtime storage

BlackBerry® devices use a runtime store as a central location in which BlackBerry Java® Applications can share runtime objects. By default, only BlackBerry Java Applications that Research In Motion digitally signs can access data in the runtime store. Contact RIM for information about how to control access to your data.

The runtime store is not persistent. When you restart the BlackBerry device, the data in the runtime store clears.

Retrieve the runtime store

1. Import the `net.rim.device.api.system.RuntimeStore` class.
2. Invoke `RuntimeStore.getRuntimeStore()`.

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
```

Add an object in the runtime store

1. Import the following classes:
 - `net.rim.device.api.system.RuntimeStore`
 - `java.lang.String`
 - `java.lang.IllegalArgumentException`
2. Invoke `RuntimeStore.put(long, String)` and provide as parameters a unique long ID and the runtime object to store.
3. Create a try - catch block to manage the `IllegalArgumentException` that `put()` throws if a runtime object with the same ID exists.

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
String msg = "Some shared text";
long ID = 0x60ac754bc0867248L;
try {
    store.put( ID, msg );
} catch(IllegalArgumentException e) {
}
```

Replace an object in the runtime store

1. Import the following classes:
 - `net.rim.device.api.system.RuntimeStore`
 - `java.lang.String`
 - `net.rim.device.api.system.ControlledAccessException`
2. Invoke `replace()`.
3. Create a try - catch block to manage the `ControlledAccessException` that `replace()` throws if the runtime object with the specified ID does not exist.

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
String newmsg = "Some new text";
try {
    Object obj = store.replace( 0x60ac754bc0867248L, newmsg);
} catch(ControlledAccessException e) {
    } not exist
```

Retrieve a registered runtime object

1. Import the following classes:
 - `net.rim.device.api.system.RuntimeStore`
 - `net.rim.device.api.system.ControlledAccessException`

2. Invoke `RuntimeStore.get()` and provide as a parameter the runtime object ID.
3. Create a try - catch block to manage the `ControlledAccessException` that `get()` throws if the BlackBerry® Java® Application does not have read access to the specified runtime object.

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
try {
    Object obj = store.get(0x60ac754bc0867248L);
} catch(ControlledAccessException e) {
}
```

Retrieve an unregistered runtime object

1. Import the following classes:
 - `net.rim.device.api.system.RuntimeStore`
 - `net.rim.device.api.system.ControlledAccessException`
 - `java.lang.RuntimeException`
2. Invoke `RuntimeStore.waitFor()` to wait for registration of a runtime object to complete.
3. Create code to handle exceptions.

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
try {
    Object obj = store.waitFor(0x60ac754bc0867248L);
} catch(ControlledAccessException e) {
} catch(RuntimeException e) {
}
```

Creating connections

3

Network gateways

Using the BlackBerry Enterprise Server as an intranet gateway

Enterprise customers host the BlackBerry® Enterprise Server behind their corporate firewall to enable access from BlackBerry devices to the corporate intranet. The BlackBerry® Mobile Data System component of the BlackBerry Enterprise Server includes the BlackBerry® MDS Services, which provides an HTTP and TCP/IP proxy service to let third-party Java® applications use it as a secure gateway for managing HTTP and TCP/IP connections to the intranet. When you use the BlackBerry Enterprise Server as an intranet gateway, all traffic between your application and the BlackBerry Enterprise Server is automatically encrypted using AES or triple DES encryption. Because the BlackBerry Enterprise Server resides behind the corporate firewall and provides inherent data encryption, applications can communicate with application servers and web servers that reside on the corporate intranet.

If your application connects to the Internet rather than to the corporate intranet, you might be able to use the BlackBerry Enterprise Server that belongs to the customer as a gateway. In this case, network requests travel behind the corporate firewall to the BlackBerry Enterprise Server, which makes the network request to the Internet through the corporate firewall. However, enterprise customers can set an IT policy to enforce that the BlackBerry Enterprise Server is the gateway for all wireless network traffic, including traffic destined for the Internet.

If your application connects to the Internet, and you are targeting non-enterprise customers, you can also use either the BlackBerry® Internet Service or the Internet gateway of the wireless server provider to manage connections.

Using the wireless service provider's Internet gateway

Java® applications for BlackBerry® devices can connect to the Internet using the Internet gateway that the wireless service provider provides. Most wireless service providers provide their own Internet gateway that offers direct TCP/IP connectivity to the Internet. Some operators also provide a WAP gateway that lets HTTP connections occur over the WAP protocol. Java applications for BlackBerry devices can use either of these gateways to establish connections to the Internet. If you write your application for BlackBerry device users who are on a specific wireless network, this approach can often yield good results. However, if you write your application for BlackBerry device users on a variety of wireless networks, testing your application against the different Internet gateways and achieving a consistent and reliable experience can be challenging. In these scenarios, you may find it useful to use the BlackBerry® Internet Service, and use the wireless service provider's Internet gateway as a default connection type if the BlackBerry Internet Service is not available.

Retrieve the wireless network name

1. Import the following classes:

- `net.rim.device.api.system.RadioInfo`
 - `java.lang.String`
 - `net.rim.device.api.ui.Field`
2. Invoke `RadioInfo.getCurrentNetworkName()`. The BlackBerry® device must be connected to a wireless network for this method to work.

```
String networkName = RadioInfo.getCurrentNetworkName();
System.out.println ("Network Name: " + networkName );
```

Connections

Use HTTP authentication

1. Import the following classes:
 - `net.rim.device.api.system.CoverageInfo`
 - `javax.microedition.io.Connector`
 - `net.rim.device.api.ui.UiApplication`
 - `net.rim.device.api.ui.component.Dialog`
 - `java.lang.String`
2. Import the following interfaces:
 - `javax.microedition.io.HttpConnection`
 - `net.rim.device.api.system.CoverageStatusListener`
 - `javax.microedition.io.StreamConnection`
3. Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to verify that the BlackBerry device is in wireless network coverage area.
4. Invoke `Connector.open()`, using the HTTP location of the protected resource.
5. Cast and store the returned object as a `StreamConnection`.

```
StreamConnection s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt");
```

6. Cast and store the `StreamConnection` object as an `HttpConnection` object.


```
HttpConnection httpConn = (HttpConnection)s;
```
7. Invoke `HttpConnection.getResponseCode()`.


```
int status = httpConn.getResponseCode();
```
8. Create code to manage an unauthorized HTTP connection attempt.

```
int status = httpConn.getResponseCode();
switch (status)
case (HttpURLConnection.HTTP_UNAUTHORIZED);
```

9. Create a `run()` method and within it implement a `dialog` object to ask the BlackBerry device user for login information.

```
UiApplication.getUiApplication().invokeAndWait(new Runnable())
{
public void run()
{
dialogResponse = Dialog.ask;
(Dialog.D_YES_NO, "Unauthorized Access:\n Do you wish to log in?");
}
}
```

10. To process the login information, create code to manage a Yes dialog response.

- a. Retrieve the login information and close the current connection.

```
if (dialogResponse == Dialog.YES)
{String login = "username:password";
//Close the connection.
s.close();
```

- b. Encode the login information.

```
byte[] encoded = Base64OutputStream.encode(login.getBytes(),
0, login.length(), false, false);
```

11. Invoke `HttpURLConnection.setRequestProperty()` using the encoded login information to access the protected resource.

```
s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt ");
httpConn = (HttpURLConnection)s;
httpConn.setRequestProperty("Authorization", "Basic " + new String(encoded));
```

Use an HTTPS connection

If your BlackBerry device is associated with a BlackBerry® Enterprise Server and uses an HTTPS proxy server that requires authentication, you will not be able to use end-to-end TLS.

1. Import the following classes:
 - `net.rim.device.api.system.CoverageInfo`
 - `javax.microedition.io.Connector`
2. Import the following interfaces:
 - `net.rim.device.api.system.CoverageStatusListener`
 - `javax.microedition.io.HttpsConnection`

- Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless network coverage area.
- Invoke `Connector.open()`, specifying HTTPS as the protocol and cast the returned object as an `HttpsConnection` object to open an HTTP connection.

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

- To specify the connection mode, add one of the following parameters to the connection string that passes to `Connector.open()`
 - Specify that an end-to-end HTTPS connection must be used from the BlackBerry device to the target server: `EndToEndRequired`
 - Specify that an end-to-end HTTPS connection should be used from the BlackBerry device to the target server. If the BlackBerry device does not support end-to-end TLS, and the BlackBerry device user permits proxy TLS connections, then a proxy connection is used: `EndToEndDesired`.

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

Use a socket connection

Although you can implement HTTP over a socket connection, you should use an HTTP connection for the following reasons:

- Socket connections do not support BlackBerry® Mobile Data System features, such as push.
- BlackBerry® device applications that use socket connections typically require significantly more bandwidth than BlackBerry device applications that use HTTP connections.

- Import the following classes:
 - `net.rim.device.api.system.CoverageInfo`
 - `javax.microedition.io.Connector`
 - `java.lang.String`
 - `java.io.OutputStreamWriter`
 - `java.io.InputStreamReader`
- Import the following interfaces:
 - `net.rim.device.api.system.CoverageStatusListener`
 - `javax.microedition.io.StreamConnection`
- Use the `CoverageInfo` class and `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless network coverage area.
- Invoke `Connector.open()`, specifying **socket** as the protocol and appending the `deviceside=false` parameter to the end of the URL.
 - To open a socket connection using BlackBerry MDS Services, append `deviceside=false` to the end of the URL. BlackBerry device applications must input their local machine IP explicitly because `localhost` is not supported.

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

- To open a socket connection over direct TCP, append the `deviceside=true` parameter to the end of the URL.

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

- To open a socket connection over direct TCP, specifying APN information, append the `deviceside=true` parameter to the end of the URL and specify the APN over which the connection will be made. Specify the user name to connect to the APN, and the password if required by the APN.

```
private static String URL = "socket://local_machine_IP:4444;deviceside=true;apn=internet.com;tunnelauthusername=user165;tunnelauthpassword=user165password";
StreamConnection conn = null;
conn = (StreamConnection)Connector.open(URL);
```

5. Use `openInputStream()` and `openOutputStream()` to send and receive data.

```
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();
};
```

6. Invoke `close()` on the input and output streams and the socket connection. Each of the `close()` methods throws an `IOException`. Make sure that the BlackBerry device application implements exception handling.

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

Use a datagram connection

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. Use a datagram connection to send and receive datagrams.

To use a datagram connection, you must have your own infrastructure to connect to the wireless network, including an APN for GPRS networks. Using UDP connections requires that you work closely with service providers. Verify that your service provider supports UDP connections.

1. Import the following classes and interfaces:
 - `net.rim.device.api.system.CoverageInfo`

- `javax.microedition.io.Connector`
 - `java.lang.String`
2. Import the following interfaces:
 - `net.rim.device.api.system.CoverageStatusListener`
 - `javax.microedition.io.DatagramConnection`
 - `javax.microedition.io.Datagram`
 3. Use the `CoverageInfo` class and the `CoverageStatusListener` interface of the `net.rim.device.api.system` package to make sure that the BlackBerry device is in a wireless network coverage area.
 4. Invoke `Connector.open()`, specify `udp` as the protocol and cast the returned object as a `DatagramConnection` object to open a datagram connection.

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

where:

- **host** is the host address in dotted ASCII-decimal format.
 - **dest-port** is the destination port at the host address (optional for receiving messages).
 - **src-port** is the local source port (optional).
 - **apn** is the network APN in string format.
5. To receive datagrams from all ports at the specified host, omit the destination port in the connection string.
 6. To open a datagram connection on a non-GPRS network, specify the source port number, including the trailing slash mark. For example, the address for a CDMA network connection would be `udp://121.0.0.0:2332;6343/`. You can send and receive datagrams on the same port.
 7. To create a datagram, invoke `DatagramConnection.newDatagram()`.


```
Datagram outDatagram = conn.newDatagram(buf, buf.length);
```
 8. To add data to a diagram, invoke `Datagram.setData()`.


```
byte[] buf = new byte[256];
outDatagram.setData(buf, buf.length);
```
 9. To send data on the datagram connection, invoke `send()` on the datagram connection.


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

If a BlackBerry®Java® Application attempts to send a datagram on a datagram connection and the recipient is not listening on the specified source port, an `IOException` is thrown. Make sure that the BlackBerry Java Application implements exception handling.
 10. To receive data on the datgram connection, invoke `receive()` on the datagram connection. The `receive()` method blocks other operations until it receives a data packet. Use a timer to retransmit the request or close the connection if a reply does not arrive.

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

- To 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 datagram connection, invoke `close()` on the input and output streams, and on the datagram connection object.

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

Use a USB or serial port connection

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

- Import the following classes:
 - `javax.microedition.io.Connector`
 - `java.io.DataOutputStream`
 - `java.lang.String`
 - `java.io.DataInputStream`
- Import the `javax.microedition.io.StreamConnection` interface.
- Invoke `Connector.open()`, and specify **comm** as the protocol and COM1 or USB as the port to open a USB or serial port connection, .

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

- To send data on the USB or serial port connection, invoke `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(data);
```

- To receive data on the USB or serial port connection, use a non-main event thread to read data from the input stream. Invoke `openInputStream()` or `openDataInputStream()`.

```
DataInputStream _din = _conn.openInputStream();
Use the read methods on the input stream to read data.
```

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

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

- To close the USB or serial port connection, invoke `close()` on the input and output streams, and on the port connection object. The `close()` method can throw `IOException`s. Make sure the BlackBerry device application implements exception handling.

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

Use a Bluetooth serial port connection

You can use the Bluetooth® API (`net.rim.device.api.bluetooth`) to let your BlackBerry® device application access the Bluetooth Serial Port Profile and initiate a server or client Bluetooth serial port connection to a computer or other Bluetooth enabled device.

- Import the following classes:
 - `javax.microedition.io.Connector`
 - `net.rim.device.api.bluetooth.BluetoothSerialPort`
 - `java.io.DataOutputStream`
 - `java.io.DataInputStream`
 - `java.lang.String`
 - `java.io.IOException`
- Import the `javax.microedition.io.StreamConnection` interface.
- Invoke `Connector.open()`, providing the serial port information that `BluetoothSerialPort.getSerialPortInfo()` returns as a parameter to open a Bluetooth connection.

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

- To send data on the Bluetooth connection, invoke `openDataOutputStream()` or `openOutputStream()`.

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

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

```
private static final int JUST_OPEN = 4;
_dout.writeInt(JUST_OPEN);
```

- To receive data on the Bluetooth connection, in a non-main event thread, invoke `openInputStream()` or `openDataInputStream()`. Use the read methods on the input stream to read the data.

```
DataInputStream _din = _bluetoothConnection.openDataInputStream();
String contents = _din.readUTF();
```

- Invoke `close()` on the input and output streams, and on the Bluetooth serial port connection object to close the Bluetooth connection. The `close()` method can throw `IOException`s. Make sure the BlackBerry device application implements exception handling.

```
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;
```

Wi-Fi connections

Wireless access families

Working with the BlackBerry® device transceiver involves using APIs that make reference to wireless access families.

Wireless access family	Description
3GPP	includes GPRS, EDGE, UMTS®, GERAN, UTRAN, and GAN
CDMA	includes CDMA1x and EVDO
WLAN	includes 802.11™, 802.11a™, 802.11b™, 802.11g™

For more information about wireless access families, see the API reference for the BlackBerry® Java® Development Environment

Retrieve the wireless access families that a BlackBerry device supports

1. Import the `net.rim.device.api.system.RadioInfo` class.
2. Invoke `RadioInfo.getSupportedWAFs()`.

Determine if a BlackBerry device supports multiple wireless access families

1. Import the `net.rim.device.api.system.RadioInfo` class.
2. Invoke `RadioInfo.areWAFsSupported(int wafs)`.

Determine the wireless access family transceivers that are turned on

1. Import the `net.rim.device.api.system.RadioInfo` class.
2. Invoke `RadioInfo.getActiveWAFs()`.

Turn on the transceiver for a wireless access family

1. Import the `net.rim.device.api.system.Radio` class.
2. Invoke `Radio.activateWAFs(int WAFs)`. The `WAFs` parameter is a bitmask.

Turn off the transceiver for a wireless access family

1. Import the `net.rim.device.api.system.Radio` class.
2. Invoke `Radio.deactivateWAFs(int WAFs)`. The `WAFs` parameter is a bitmask.

Check if the Wi-Fi transceiver is turned on

1. Import the `net.rim.device.api.system.RadioInfo` class.
2. Create an IF statement that tests the value of `RadioInfo.WAF_WLAN` and the value returned by `RadioInfo.getActiveWAFs()`.

```
if ( ( RadioInfo.getActiveWAFs() & RadioInfo.WAF_WLAN ) != 0 ) { ... }
```

Check if the Wi-Fi transceiver is connected to a wireless access point

1. Import the `net.rim.device.api.system.WLANInfo` class.
2. Create an IF statement that tests the value of `WLANInfo.WLAN_STATE_CONNECTED` and the value returned by `WLANInfo.getWLANState()`. The `WLANInfo.getWLANState()` method checks if a BlackBerry® device has an IP address and can transfer data over a Wi-Fi® network. If the transceiver for the WLAN wireless access family is off, this method returns `WLANInfo.WLAN_STATE_DISCONNECTED`.

```
if (WLANInfo.getWLANState() == WLANInfo.WLAN_STATE_CONNECTED) {...}
```

Retrieve the status of the wireless access point or the active Wi-Fi profile

You can let a BlackBerry® device application retrieve status information such as the data rate of the connection, the wireless LAN standards used (802.11a™, 802.11b™, or 802.11g™), the SSID of the associated access point, or the name of the Wi-Fi profile in use. The transceiver for the WLAN wireless access family must be connected to a wireless access point.

1. Import the `net.rim.device.api.system.WLANInfo` class.
2. Invoke `WLANInfo.getAPInfo()`, storing a reference to `WLANInfo.WLANAPIInfo` that this method returns. The `WLANInfo.WLANAPIInfo` object contains a snapshot of the current wireless network.

```
WLANInfo.WLANAPIInfo info = WLANInfo.getAPInfo();
```

If the BlackBerry device is not connected to an access point, the `WLANInfo.getAPInfo()` method returns null.

Open a Wi-Fi socket connection

The `interface=wifi` parameter applies only to TCP/UDP connections. To establish a Wi-Fi® connection and use a Wi-Fi API in a BlackBerry® device application, the wireless service provider must support Wi-Fi access.

1. Import the following classes:
 - `java.lang.String`
 - `javax.microedition.io.Connector`
2. Import the `javax.microedition.io.StreamConnection` interface.
3. Invoke `Connector.open()`, specify **socket** as the protocol, and append the `deviceside=true` parameter and the `interface=wifi` parameter to the end of the URL string value.

```
private static String URL = "socket://local_machine_IP:  
4444;deviceside=true;interface=wifi";  
StreamConnection conn = null;  
conn = (StreamConnection)Connector.open(URL);
```

Open a Wi-Fi HTTP connection

The `interface=wifi` parameter applies only to TCP/UDP connections. To establish a Wi-Fi® connection and use a Wi-Fi API in a BlackBerry® device application, the wireless service provider must support Wi-Fi access.

1. Import the following classes:
 - `java.lang.String`
 - `javax.microedition.io.Connector`
2. Import the `javax.microedition.io.HttpConnection` interface.

3. Invoke `Connector.open()`, specify **http** as the protocol, and append the `interface=wifi` parameter to the end of the URL string value.
4. Cast the returned object as an `HttpConnection` or a `StreamConnection` object.

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

Open a Wi-Fi HTTPS connection

The `interface=wifi` parameter applies only to TCP/UDP connections. To establish a Wi-Fi® connection and use a Wi-Fi API in a BlackBerry® device application, the wireless service provider must support Wi-Fi access.

1. Import the following classes:
 - `java.lang.String`
 - `javax.microedition.io.Connector`
2. Import the `javax.microedition.io.HttpsConnection` interface.
3. Invoke `Connector.open()`, specify **https** as the protocol, and append the `interface=wifi` parameter to the end of the URL string value.
4. Cast the returned object as an `HttpsConnection` object.

```
HttpsConnection conn = null;  
String URL = "https://host:443/;deviceside=true;interface=wifi";  
conn = (HttpsConnection)Connector.open(URL);
```

Managing applications

4

Application manager

The BlackBerry® Java® Virtual Machine on the BlackBerry® device includes an application manager that functions as the central dispatcher of operating system events for other BlackBerry device applications.

The `net.rim.device.api.system.ApplicationManager` class lets BlackBerry device applications interact with the application manager to perform the following actions:

- interact with processes, such as retrieving the IDs for foreground BlackBerry device applications
- post global events to the system
- run a BlackBerry device application immediately or at a specific time

Retrieve information about a BlackBerry Java Application

1. Import the following classes:
 - `net.rim.device.api.system.ApplicationManager`
 - `net.rim.device.api.system.ApplicationDescriptor`
 - `java.lang.String`
 - `net.rim.device.api.system.`

2. To retrieve information about the processes that are running, invoke `ApplicationManager.getVisibleApplications()`.

```
ApplicationManager manager = ApplicationManager.getApplicationManager();
ApplicationDescriptor descriptors[] = manager.getVisibleApplications();
```

3. To retrieve descriptions of the objects for the BlackBerry® device applications that are running, invoke `ApplicationDescriptor.getName()`.

```
String appname1 = descriptors[0].getName();
```

4. To retrieve a description of the current BlackBerry device application, invoke `ApplicationDescriptor.currentApplicationDescriptor()`

```
ApplicationDescriptor descriptor =
ApplicationDescriptor.currentApplicationDescriptor();
```

Communicate with another BlackBerry Java Application

1. Import the `net.rim.device.api.system.ApplicationManager` class.

2. To post a system-level event to another BlackBerry® device application, invoke one of the `ApplicationManager.postGlobalEvent()` methods.

Determine the services that are available to a BlackBerry Java Application

The service book consists of service records. Each service record defines a service on a BlackBerry® device. Service records define the communication protocol (WAP or IPPP), the network gateway, and the configuration information such as the browser settings.

1. Import the `net.rim.device.api.servicebook` class.
2. To let your BlackBerry device application interact with the BlackBerry® Infrastructure, use the service book API (`net.rim.device.api.servicebook`).

Application control

The BlackBerry® Application Control IT policy rules provide administrators with the ability to establish the capabilities of an application when it runs on a specific BlackBerry device. For example, administrators can use the BlackBerry Application Control IT policy rule to make sure that a game that exists on the BlackBerry device cannot access the phone API. The BlackBerry Application Control IT policy rule works only when the BlackBerry device is connected to a BlackBerry® Enterprise Server. This IT policy does not apply to BlackBerry devices that use the BlackBerry® Internet Service only.

If the administrator or a user denies the application access to one of the protected areas, the associated method throws a `ControlledAccessException`. For class-level checks, the method throws a `NoClassDefFoundError`. Depending on which APIs that you use, your application might need to handle both types of errors.

Allow a BlackBerry device application to request access to resources

1. Import the following classes:
 - `net.rim.device.api.applicationcontrol.ApplicationPermissions`
 - `net.rim.device.api.applicationcontrol.ApplicationPermissionsManager`
2. Create an instance of the `ApplicationPermissions` class.

```
ApplicationPermissions permissions = new ApplicationPermissions();
```

3. Specify the build request to ask for event injection privileges.

```
permissions.addPermission( ApplicationPermissions.PERMISSION_EVENT_INJECTOR );
```

4. Determine the access control settings that the BlackBerry® device user specifies.

```
if( ApplicationPermissionsManager.getInstance().invokePermissionsRequest  
( permissions ) ) {  
    System.out.println( "The user saved equal, or more permissive settings" );  
}
```

```
} else {  
System.out.println( "The user saved more restrictive settings" );  
}
```

Code modules

Retrieve module information

1. Import the `net.rim.device.api.system.CodeModuleManager` class.
2. To retrieve a handle for a module, invoke `getModuleHandle()`, and provide the name of the code module as a parameter.

```
int handle = CodeModuleManager.getModuleHandle("test_module")
```

3. To retrieve specific information about a module, invoke the methods of the `CodeModuleManager` class, and provide the module handle as a parameter to these methods.

```
String name = CodeModuleManager.getModuleName( handle );  
String vendor = CodeModuleManager.getModuleVendor( handle );  
String description = CodeModuleManager.getModuleDescription( handle );  
int version = CodeModuleManager.getModuleVersion( handle );  
int size = CodeModuleManager.getModuleCodeSize( handle );  
int timestamp = CodeModuleManager.getModuleTimestamp( handle );
```

4. To retrieve an array of handles for existing modules on a BlackBerry® device, invoke `getModuleHandles()`.

```
int handles[] = CodeModuleManager.getModuleHandles();  
String name = CodeModuleManager.getModuleName( handles[0] );
```

Access control messages

Displaying a message for an operation that requires user permission

You can use the components of the `net.rim.device.api.applicationcontrol` package to let an BlackBerry® device application display custom messages to a BlackBerry device user when the BlackBerry device application attempts an operation that the BlackBerry device user must permit. The BlackBerry device application displays information about the type of permission that the user must provide. For example, you can use `PERMISSION_PHONE` for an operation that requires access to the phone functionality of the BlackBerry device.

You can use the `applicationcontrol` package to include a custom message with the default message that a BlackBerry device application displays in response to an application control.

Display an application control message to a user

A BlackBerry® device application can include more than one registered `ReasonProvider`. The BlackBerry device application displays messages from `ReasonProviders` in the order that each `ReasonProvider` registers with the BlackBerry device application. For example, if a BlackBerry device application registers `ReasonProvider A` before `ReasonProvider B`, the BlackBerry device application displays the message from `ReasonProvider A`, followed by the message from `ReasonProvider B`.

1. Import the `net.rim.device.api.applicationcontrol.ReasonProvider` interface:
2. In your implementation of the `ReasonProvider.getMessage(int permissionID)` method, return a `String` value that contains the message to display to a BlackBerry user.

Using custom messages and folders in the message list

5

You can use the `net.rim.blackberry.api.messageList` package to create custom messages and folders for a BlackBerry® device application.

To use custom folders and messages in a BlackBerry device application, you must create an application with the following modules:

- a UI module for interacting with a BlackBerry device user
- a module for interacting with an application server and performing other actions in the background

Both modules are part of the same BlackBerry device application but have different application entry points.

You must determine the functionality that is part of each module. For example, a menu item that lets a BlackBerry device user delete a message or mark a message as read should be part of the daemon module. A menu item that lets a BlackBerry device user open or reply to a message should be part of the UI module.

Creating a module for background processes

The daemon module runs without input from a BlackBerry® device user and can send messages to and receive messages from an application server and can add messages to the global message list. The daemon module can transfer messages using a native protocol or through email messaging. The daemon module starts automatically when the BlackBerry device starts. A BlackBerry device user cannot start or stop the daemon module.

The daemon module can also register an BlackBerry device application folder when the BlackBerry device starts and can listen for updates to the application message folder such as when a BlackBerry device user deletes a message or marks a message as read or unread.

Creating a module for the UI

The UI module runs when a BlackBerry® device user clicks the icon for a BlackBerry device application from the Home screen and is used to interact with a BlackBerry device user. The UI module can contain a dialog box, a screen, fields, and other components.

For example, if a BlackBerry device user highlights a custom message in the global message list and clicks the trackball, the UI module should start and render the message content. If the UI module is not running and a BlackBerry device user attempts to open custom message, the UI module starts automatically. The UI module can also provide Reply or Compose message functions.

Create the module for background processes

1. Create a project as **Auto-Run On Startup**.

2. Create a class with the fields and methods for the module.
3. Compile the project into a .jad file.
4. Include the .jad file with the .jad files for the UI module and the main part of the program.

Start the module for background processes or the module for the UI

1. Import the `net.rim.blackberry.api.messageList.ApplicationMessageFolderRegistry` class.
2. Create a main method for the BlackBerry® device application.

```
public static void main( String[] args )
{
    try {
```

3. In the `main()` method, check if the value of the `args` parameter indicates that the BlackBerry device application should start the daemon module.

```
if( args.length == 1 && args[ 0 ].equals( "daemon" ) ) {
```

4. Create an instance of a class that contains the daemon functionality and items.

```
MLSampleDaemon daemon = new MLSampleDaemon();
```

5. Obtain a reference to the `ApplicationMessageFolderRegistry`.

```
ApplicationMessageFolderRegistry reg =
ApplicationMessageFolderRegistry.getInstance();
```

6. In the `main()` method, check if the value of the `args` parameter indicates that the BlackBerry device application should start the UI module.

```
} else if( args.length == 1 && args[ 0 ].equals( "gui" ) ) {
```

7. Create an instance of a class that contains the UI functionality and items.

```
MLSampleGui gui = new MLSampleGui();
```

8. Display the UI for the BlackBerry device application .

```
gui.showGui();
```

9. Add the application to the event dispatcher.

```
gui.enterEventDispatcher();
```

Create an icon for a custom message

You can associate an icon with a message. In the message list, the icon displays on the left side of a message.

1. Import the following classes:

- `net.rim.device.api.system.EncodedImage`
 - `net.rim.blackberry.api.messageList.ApplicationIcon`
 - `net.rim.blackberry.api.messageList.ApplicationMessageFolderRegistry`
2. Import the `net.rim.blackberry.api.messageList.ApplicationMessage` interface.
 3. To create an icon based on an encoded image, when you create an instance of an `ApplicationIcon`, invoke `EncodedImage.getEncodedImageResource` with the name of the image file as an argument.

```
ApplicationIcon newIcon = new ApplicationIcon
( EncodedImage.getEncodedImageResource( "ml_sample_new.png" ) );
ApplicationIcon openedIcon = new ApplicationIcon
( EncodedImage.getEncodedImageResource( "ml_sample_opened.png" ) );
```

4. To assign a status and an icon to a message, invoke `ApplicationMessageFolderRegistry.registerMessageIcon` and specify the following as parameters: a value for the message type for an BlackBerry® device application, a field from the `ApplicationMessage.Status` interface as the status argument, and an instance of an `ApplicationIcon`.

```
int MESSAGE_TYPE = 0;
reg.registerMessageIcon( MESSAGE_TYPE, STATUS_NEW, newIcon );
reg.registerMessageIcon( MESSAGE_TYPE, STATUS_OPENED, openedIcon );
```

Create a custom folder in the message list

To be able to perform operations on custom messages, the BlackBerry® device application must register at least one custom folder.

1. Import the following classes:
 - `net.rim.blackberry.api.messageList.ApplicationMessageFolderRegistry`
 - `net.rim.blackberry.api.messageList.ApplicationMessageFolder`
2. Import the following interfaces:
 - `net.rim.blackberry.api.messageList.ApplicationMessage`
 - `net.rim.device.api.collection.ReadableList`
 - `net.rim.blackberry.api.messageList.ApplicationMessageFolderListener`
3. Create a class that implements the `ApplicationMessage` interface.

```
public class MLSampleMessage implements ApplicationMessage
```

4. Obtain a reference to the `ApplicationMessageFolderRegistry`.

```
ApplicationMessageFolderRegistry reg =
ApplicationMessageFolderRegistry.getInstance();
```

5. Register an BlackBerry device application folder for each collection of messages.

```

ReadableList inboxMessages = messages.getInboxMessages(); // collection with
MLSAMPLEMESSAGE elements
ReadableList deletedMessages = messages.getDeletedMessages(); // collection with
MLSAMPLEMESSAGE elements
ApplicationMessageFolder inboxFolder = reg.registerFolder( INBOX_FOLDER_ID,
    "Inbox", inboxMessages );
ApplicationMessageFolder deletedFolder = reg.registerFolder( DELETED_FOLDER_ID,
    "Deleted Messages",
    deletedMessages, false );

```

- Let an BlackBerry device application be notified when specific folder events occur.

```

deletedFolder.addListener( this ,
    ApplicationMessageFolderListener.MESSAGE_DELETED );

```

- Create a class that implements the `ApplicationMessageFolderListener` interface.

```

public class AppFolderListener implements ApplicationMessageFolderListener

```

- To let an BlackBerry device application perform actions when a folder event occurs, implement the `actionPerformed()` method of the `ApplicationMessageFolderListener` interface.

```

public void actionPerformed( int action, ApplicationMessage[] messages,
    ApplicationMessageFolder folder ) {
    // check if action was performed on multiple messages
    if( messages.length == 1 ) {
        switch( action ) {
            case ApplicationMessageFolderListener.MESSAGE_DELETED:
                messageStore.deleteInboxMessage( message );
        }
    }
}

```

- Set the root folder for the folders of the BlackBerry device application. The name of the root folder appears in the View Folder dialog of the Message list application when a BlackBerry device application registers more than one application message folder.

```

reg.setRootFolderName( "ML Sample" );

```

Send a notification when a custom folder changes

- Import the `net.rim.blackberry.api.messageList.ApplicationMessageFolder` class.
- To notify a BlackBerry® device application when a message is added to a custom folder, invoke `ApplicationMessageFolder.fireElementAdded()`.

```

inboxFolder.fireElementAdded( newMessage );

```

- To notify a BlackBerry device application when a message is removed from a custom folder, invoke `ApplicationMessageFolder.fireElementRemoved()`.

```

inboxFolder.fireElementRemoved( deletedMessage );

```

4. To notify a BlackBerry device application when a message in a custom folder is updated, invoke `ApplicationMessageFolder.fireElementUpdated()`.

```
inboxFolder.fireElementUpdated( updatedMessage );
```

5. To notify a BlackBerry device application when more than one message in a custom folder changes, invoke `ApplicationMessageFolder.fireReset()`.

```
inboxFolder.fireReset();
```

Create an indicator for the number of messages in a custom folder

1. Import the following classes:
 - `net.rim.blackberry.api.messageList.ApplicationIndicatorRegistry`
 - `net.rim.device.api.system.EncodedImage`
 - `net.rim.blackberry.api.messageList.ApplicationIcon`
 - `net.rim.blackberry.api.messageList.ApplicationIndicator`
2. Store a reference to an `ApplicationIndicatorRegistry` in an `ApplicationIndicatorRegistry` variable.

```
ApplicationIndicatorRegistry reg = ApplicationIndicatorRegistry.getInstance();
```

3. To create an encoded image from an image, invoke `EncodedImage.getEncodedImageResource` using the name of the image file as an argument. Save a reference to the encoded image in an `EncodedImage` variable.

```
EncodedImage image = EncodedImage.getEncodedImageResource( "clouds.gif" );
```

4. To create an BlackBerry® device application icon based on the encoded image, create an instance of an `ApplicationIcon` using an `EncodedImage` as an argument.

```
ApplicationIcon icon = new ApplicationIcon( image );
```

5. Use an icon with an application indicator.

```
ApplicationIndicator indicator = reg.register( icon, false, true);
```

6. To retrieve the indicator that the BlackBerry device application registered, invoke `ApplicationIndicatorRegistry.getApplicationIndicator()` and store the return value in an `ApplicationIndicator` variable.

```
ApplicationIndicator AppIndicator = reg.getApplicationIndicator();
```

7. To set the icon and value of an indicator, invoke `ApplicationIndicator.set()`.

```
AppIndicator.set( newIcon, newValue );
```

Hide an indicator for a custom folder

1. Import the `net.rim.blackberry.api.messageList.ApplicationIndicator` class.
2. To temporarily hide the indicator, invoke `ApplicationIndicator.setVisible()`.

```
OldIndicator.setVisible( false );
```

Remove an indicator for a custom folder

1. Import the `net.rim.blackberry.api.messageList.ApplicationIndicatorRegistry` class.
2. To unregister an indicator, invoke `ApplicationIndicatorRegistry.unregister()`.

```
reg.unregister
```

Applications for push content

6

Types of push applications

Push applications send web content or data to specific BlackBerry® device users. Users do not need to request or download the data because the push application delivers the information as soon as it becomes available.

Application	Description
browser push applications	<p>Browser push applications send content to a web browser on the BlackBerry device.</p> <ul style="list-style-type: none"> The BlackBerry® Browser configuration supports BlackBerry MDS Services push applications. The WAP Browser configuration supports WAP push applications. The Internet Browser configuration does not support push applications. <p>For more information about creating browser push applications, see the <i>BlackBerry Browser Developer Guide</i>.</p>
client/server push applications	<p>Client/server push applications consist of a custom client BlackBerry device application on the BlackBerry device and a server-side application that pushes content to the client BlackBerry device application. This approach provides more control than browser push applications over the type of content that you can send and how the BlackBerry device processes and displays the content.</p>

Types of push requests

The BlackBerry® Mobile Data System up to 1000 push requests, including both RIM and PAP push requests.

BlackBerry® device applications can send the following types of push requests:

Request	Supported tasks	Push storage
RIM push	<ul style="list-style-type: none"> sending a server-side push submission specifying a reliability mode for the push submission 	<p>RIM push requests are stored in RAM.</p> <p>Undelivered RIM push requests might be lost if the server restarts.</p>

Request	Supported tasks	Push storage
	<ul style="list-style-type: none"> • specifying a deliver-before time stamp for the push submission • requesting a result notification of the push submission • specifying a deliver-after time stamp for the push submission 	
PAP	<ul style="list-style-type: none"> • sending a server-side push submission • specifying a reliability mode for the push submission • specifying a deliver-before time stamp for the push submission • requesting a result notification of the push submission • specifying a deliver-after time stamp for the push submission • cancelling a push request submission • querying the status of a push request submission <p>For more information about PAP, visit www.openmobilealliance.org</p>	PAP push requests are stored in a database.

Localizing BlackBerry device applications

7

Multilanguage support

The BlackBerry® Integrated Development Environment includes a resource mechanism for creating string resources. The Localization API is part of the `net.rim.device.api.i18n` package. MIDP applications do not support localization.

The BlackBerry Integrated Development Environment stores resources for a locale 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 locale of the BlackBerry device user, without requiring new resource bundles.

You can use the BlackBerry Integrated Development Environment to compile each resource bundle into a separately compiled `.cod` file. You can load the appropriate `.cod` files onto BlackBerry devices with the other `.cod` files for the application.

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.

Files required for localization

File required for localization	Description	Example
Resource header file	This file defines descriptive keys for each localized string. When the BlackBerry® Integrated Development Environment builds a project, it creates a resource interface with <code>Resource</code> appended to the <code>.rrh</code> file name. For example, if you create <code>AppName.rrh</code> , the interface is named <code>AppNameResource</code> .	<code>AppName.rrh</code>
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.	<code>AppName.rrc</code>
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 (<code>_</code>) and the language code, and then, optionally, an underscore (<code>_</code>) and country code.	<code>AppName_en.rrc</code> <code>AppName_en_GB.rrc</code> <code>AppName_fr.rrc</code>

File required for localization	Description	Example
	Save resource content files in the folder where the .java file is located. For example, in the folder that contains CountryInfo.java, save CountryInfo.rrc (root locale), CountryInfo_en.rrc (English), and CountryInfo_fr.rrc (French).	
Initialization file	This file initializes the resource bundle mechanism. You require this file only when you compile resources as a separate project.	init.java

Manage localization files for a suite of BlackBerry device applications

If you create a suite of BlackBerry® device applications, organize resources into separate projects for each locale. The BlackBerry® Integrated Development Environment provides a built-in initialization mechanism. You only need to create an empty initialization class with an empty `main()`. 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 the library, define a dependency between the projects.

1. Open the BlackBerry® Integrated Development Environment.
2. Create a project for each resource bundle (locale), including the root locale.
3. Give the projects for each resource 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**, and **com_company_app__fr**.
4. Right-click the project, and then click **Properties**.
5. On the **Build** tab, in the **Output file name** field, type a name for the compiled file, without a file name extension.
6. Create an initialization file.

```
package com.rim.samples.device.resource;
import net.rim.device.api.i18n.*;
public class init {
public static void main (String[] args) { }
}
```

7. Create one resource header file for each BlackBerry device application.
8. Copy the resource header (.rrh) files to the project for each BlackBerry device application.
9. Copy the resource header files to each resource locale project.
10. Create one resource content file for each BlackBerry device application.
11. Create one resource content file for each supported locale.
12. In each resource locale project, right-click each .rrh file, and then click **Properties**.
13. Select **Dependency only. Do not build**.

14. Add the resource content (.rrc) files to the projects for the appropriate locales.

Controlling access to APIs and application data

8

Check if a code signature is required

Research In Motion tracks the use of sensitive APIs in the BlackBerry® Java® Development Environment for security and export control reasons.

Locate the item in the API reference for the BlackBerry Java Development Environment. If the item has a lock icon or is noted as 'signed' your BlackBerry device application requires a signed key or signature, which RIM provides, before you can load the BlackBerry device application .cod files onto a BlackBerry device.

Java APIs with controlled access

RIM controls Runtime APIs, BlackBerry® Application APIs, and BlackBerry Cryptography APIs.

You can test BlackBerry device applications that use controlled APIs in the BlackBerry® Smartphone Simulator without code signatures; however, you must obtain code signatures from RIM before you can load the BlackBerry device applications onto BlackBerry devices.

If you use any of the following BlackBerry API packages, your BlackBerry device application requires code signatures before you can load it onto a BlackBerry device:

- `net.rim.blackberry.api.browser`
- `net.rim.blackberry.api.invoke`
- `net.rim.blackberry.api.mail`
- `net.rim.blackberry.api.mail.event`
- `net.rim.blackberry.api.menuitem`
- `net.rim.blackberry.api.options`
- `net.rim.blackberry.api.pdap`
- `net.rim.blackberry.api.phone`
- `net.rim.blackberry.api.phone.phonelogs`
- `net.rim.device.api.browser.field`
- `net.rim.device.api.browser.plugin`
- `net.rim.device.api.crypto`
- `net.rim.device.api.io.http`
- `net.rim.device.api.notification`
- `net.rim.device.api.servicebook`
- `net.rim.device.api.synchronization`
- `net.rim.device.api.system`

Register to use controlled APIs

1. Complete the registration form at <https://www.blackberry.com/SignedKeys/>.
2. Save the .csi file that Research In Motion sends to you in an email message. The .csi file contains a list of signatures and your registration information. If the BlackBerry® Signing Authority Tool administrator does not provide you with the .csi file or the Client PIN and you are an ISV partner, contact your ISV Technical Partnership Manager. If you are not an ISV partner, send an email message to jde@rim.com.
3. Double-click the .csi file.
4. If a dialog box appears that states that a private key cannot be found, complete steps 5 to 8 before you continue. Otherwise, proceed to step 9.
5. Click **Yes** to create a new key pair file.
6. In the **Private Key Password** field, type a password of at least eight characters, and type it again to confirm. The private key password protects your private key. If you lose this password, you must register again with RIM. If this password is stolen, contact RIM immediately.
7. Click **OK**.
8. Move your mouse to generate data for a new private key.
9. In the **Registration PIN** field, type the **PIN** that RIM provided.
10. In the **Private Key Password** field, type the **private key** password.
11. Click **Register**.
12. Click **Exit**.

Restrictions on code signatures

The BlackBerry® Signing Authority Tool administrator might place restrictions on your .csi file to limit your access to code signatures. To request changes to these restrictions, contact your administrator.

.csi file restriction	Description
number of requests	<p>This restriction specifies the maximum number of requests you can make using a particular .csi file. When you reach the maximum number of requests, the .csi file becomes invalid. To make new code signature requests, you must apply for a new .csi file.</p> <p>Although an administrator can permit an infinite number of requests, the number of requests is often specified to be a finite number for security reasons.</p>

.csi file restriction	Description
expiry date	This restriction specifies the expiry date for your .csi file. After the expiry date, you can no longer apply for code signatures with this .csi file. To make new signature requests, you must apply for a new .csi file.

Request a code signature

The BlackBerry® Signature Tool is included in the BlackBerry® Java® Development Environment. The BlackBerry JDE is available for download at www.blackberry.com/developers. The Web Signer application is installed when you install the BlackBerry® Signing Authority Tool.

For more information about the Web Signer application, see the BlackBerry Signing Authority Tool version 1.0 - Password Based Administrator Guide.

Before you begin: You must obtain a .csi file from Research In Motion.

1. In Windows® Internet Explorer®, locate the .cod file for the BlackBerry device application for which you are requesting a signature.
2. Make sure that a .csi file with the same name as the .cod file exists in the same folder as the .cod file. The BlackBerry® Integrated Development Environment compiler automatically generates the .csi file.
3. Double-click the .cod file to add it to the signature list. The signature list contains information on the .cod files that you want permission to access and are requesting signatures for.
4. Repeat steps 1 through 3 for each .cod file that you want to add to the signature list.
5. On the BlackBerry Signature Tool menu, click **Request**.
6. In the dialog box, type your private key password.
7. Click **OK**.

The BlackBerry Signature Tool uses the private key password to append the signature to the request, and it sends the signature list of .cod files to the Web Signer application for verification.

Register a signature key using a proxy server

You can register each .csi file only once.

1. At the command prompt, navigate to the BlackBerry® Signature Tool bin directory. For example:
`C:\Program Files\Research In Motion\BlackBerry JDE 4.6.0\bin`
2. Type
`Java -jar -Dhttp.proxyHost=myproxy.com-Dhttp.proxyPort=80SignatureTool.jar SigKey.csi` with the fol

- `SigKey`: The name of each signature key (.csi) file. Use the following naming conventions for the keys: `client-RRT-*.csi`, `client-RBB-*.csi`, `client-RCR-*.csi`.
 - `Dhttp.proxyHost`: The name or IP address of the proxy server.
 - `Dhttp.proxyPort`: The proxy server port number if you do not specify 80 as the default port number.
3. Repeat step 2 for each .csi file that you want to register.

Sign an application using a proxy server

Your registration key and .csk file are stored together. If you lose the registration key or the .csk file, you cannot request code signatures. If you are not an ISV partner, contact your ISV Technical Partnership Manager. If you are a non ISV partner, send an email message to jde@rim.com.

1. At the command prompt, navigate to the BlackBerry® Signature Tool bin directory. For example:
`C:\ProgramFiles\ResearchInMotion\BlackBerryJDE4.6.0\bin`
2. Type `Java -jar -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80 SignatureTool.jar`
3. In the File Selection window, select the .cod file(s) to sign.
4. Click **Open**.

View the signature status for an application

For files that are not signed, the **Status** column contains **Failed**. The Web Signer might have rejected the .cod file because the private key password was typed incorrectly.

1. Start the BlackBerry® Signature Tool.
2. Select a .cod file.
3. View the **Status** column.

For files the Web Signer has signed, the **Status** column contains **Signed**.

Using keys to protect APIs and data

To create an internal key pair to use with an internal signing authority system, or an external key pair to use with an external signing authority system, you must apply protection to sensitive APIs, protect data in the runtime store, and protect data in a persistent object.

The `RSAE.key` is an external key, and `ACMI.key` is an internal key.

Protect APIs using code signing keys

1. After you receive an internal key, an external key, or both keys, in the BlackBerry® Integrated Development Environment, open the project that contains the APIs that you want to control access to.
2. In the Workspace window, right-click the project file.
3. Click **Add File to Project**.
4. In the **Look In** field, browse to C:\Program Files\Research In Motion\BlackBerry Password Based Code Signing Authority\data or the location where the .key file is saved.
5. Perform one of the following tasks:

Option	Description
Use an internal key	<ol style="list-style-type: none"> a. Select the internal .key file, for example, the ACMI.key file. b. Click Open. c. In the Workspace window, double click the .key file. d. Select the Use as default for public classes option and Use as default for non-public classes settings. e. Click OK. f. In the Workspace window, right-click the project file. g. Click Add File to Project. h. In the Look In field, browse to C:\Program Files\Research In Motion\BlackBerry Password Based Code Signing Authority\data.
Use an external key	<ol style="list-style-type: none"> a. Select the external .key file, for example, the RSAE.key file. b. Click Open. c. In the Packages and classes protection window, find the name of the package that contains the sensitive API items. d. Expand the package contents. e. Select each API element that requires access control.

6. Click **Ok**.
7. Re-compile the project.

Protect runtime store data using code signing keys

1. Import the following classes:
 - `java.util.Hashtable`

- `net.rim.device.api.system.RuntimeStore`
2. Create a hash ID for the object you want to store in the runtime store.

```
long MY_DATA_ID = 0x33abf322367f9018L;
Hashtable myHashtable = new Hashtable();
```
 3. Store the object in the runtime store and protect the object with the `CodeSigningKey` object. Only applications signed with the key can read or change the object.

```
RuntimeStore.put( MY_DATA_ID, new ControlledAccess( myHashtable, key ) );
```
 4. Make sure that the object is protected with a particular code signing key, invoke `RuntimeStore.get`, providing as parameters the hash ID for the object and the `CodeSigningKey` object.

Protect persistent data using code signing keys

1. Import the following classes:
 - `java.util.Hashtable`
 - `net.rim.device.api.system.PersistentObject`
2. Create a hash ID for the object you want to store in a persistent object.

```
long MY_DATA_ID = 0x33abf322367f9018L;
Hashtable myHashtable = new Hashtable();
```
3. Store the object in the persistent object and protect the object with the `CodeSigningKey` object. For example, after a BlackBerry device application runs the following line of code, only code files that are signed with the `RSAE.key` file can read or overwrite the object in the persistent object.

```
persistentObject.setContents( new ControlledAccess( myHashtable, key ) );
```
4. Make sure that the object is protected, invoke `getContents` using the `CodeSigningKey` object as a parameter.

```
Hashtable myHashtable = (Hashtable) persistentObject.getContents( key );
```

Testing a BlackBerry device application

9

Testing applications on a BlackBerry Smartphone Simulator

After you develop and compile your application, you can test it on the BlackBerry® device. The most common first step is to set the BlackBerry® Java® Development Environment to use a BlackBerry® Smartphone Simulator. The BlackBerry Smartphone Simulator runs the same Java code as the BlackBerry devices, so the BlackBerry Smartphone Simulator provides an accurate environment for testing how applications will function on a BlackBerry device. The BlackBerry JDE includes current versions of the BlackBerry Smartphone Simulator. To download additional versions of the BlackBerry Smartphone Simulator, visit www.blackberry.com/developers/index.shtml.

Testing applications on a BlackBerry device

After you test your application on the BlackBerry® Smartphone Simulator, you can install your application on a BlackBerry device. If your application uses signed APIs, you might need code signing keys. After you install the application on the BlackBerry device, you can open the application and test its functionality and performance.

For debugging purposes, you can attach your device to the BlackBerry® Integrated Development Environment and use the debugging tool to step through your application code. The BlackBerry IDE can be useful if you are trying to identify a network or Bluetooth® issue, or other issues that are difficult to simulate.

Testing applications using the compiled .cod files

When you build a project using the BlackBerry® Integrated Development Environment, the BlackBerry IDE compiles your source files into Java® bytecode, performs preverification, and creates a single .cod file and .jad file for a BlackBerry® device application.

If a BlackBerry device application contains more than 64 KB of bytecode or resource data, the BlackBerry IDE creates a .cod file that contains sibling .cod files. Only the BlackBerry® Browser supports wireless installation of a .cod file that contains sibling .cod files. To determine if a .cod file contains sibling .cod files, extract the contents of the .cod file. Any .cod files within the original .cod file are the sibling files.

To identify modules that a BlackBerry device application requires, but are not provided with it, examine the Java® application descriptor (.jad) file RIM-COD-Module-Dependencies attribute.

Install and remove a .cod file for testing

To load, remove, or save .cod files when testing a BlackBerry® device application, use the JavaLoader tool, included with the BlackBerry® Java® Development Environment. For production applications, use the BlackBerry® Desktop Software. You must load BlackBerry device applications with dependencies in the correct order. If project A is dependent on project B, load the project B .cod file before loading the project A .cod file.

Save a .cod file from a device to a computer

To load, remove, or save .cod files when testing a BlackBerry® device application, use the JavaLoader tool, included with the BlackBerry® Java® Development Environment.

1. Connect the BlackBerry device to the computer.
2. Open a command prompt, and navigate to the location of the JavaLoader.exe file.
3. Perform one of the following actions:

Task	Steps
Save a BlackBerry device application .cod file from the BlackBerry device to your computer.	Issue a command using the following format: <code>java loader save .cod file</code> For example: <code>java loader .exe save MyApplication.cod</code>
Save BlackBerry device application .cod files listed in the same .jad file from the BlackBerry device to your computer.	Issue a command using the following format: <code>java loader save .jad file</code> For example: <code>java loader .exe save MyApplication.jad</code>
Save BlackBerry device application .cod files stored in the same CodeModuleGroup from the BlackBerry device to your computer.	Issue a command using the following format: <code>java loader save [-g] module</code> For example: <code>java loader .exe save -g MyApplication</code>

Retrieve information about a .cod file

To load, remove, or save .cod files when testing a BlackBerry® device application, use the JavaLoader tool, included with the BlackBerry® Java® Development Environment.

1. Connect the BlackBerry® device to the computer.
2. Open a command prompt and navigate to the location of the JavaLoader.exe file.
3. Perform one of the following actions:

Task	Steps
Retrieve the Name, Version, Size, and Date created information for a .cod file.	Issue a command using the following format: <code>java loader info .cod file</code> For example: <code>java loader .exe info MyApplication.cod</code>
Retrieve a list of .cod files that a .cod file requires to run.	Issue a command using the following format: <code>java loader info [-d] .cod file</code> For example: <code>java loader .exe info -d MyApplication.cod</code>
Retrieve information on <ul style="list-style-type: none">• sibling .cod files• size of code section• size of data section• size of initialized data• number of class definitions• list of signatures applied to a .cod file	Issue a command using the following format: <code>java loader info [-v] .cod file</code> For example: <code>java loader .exe info -v MyApplication.cod</code>

Packaging and distributing a BlackBerry Java Application 10

Preverify a BlackBerry device application

To reduce the amount of processing the BlackBerry® device performs when you load your BlackBerry device application, partially verify your classes. You may also use the BlackBerry® Smartphone Simulator to preverify .cod files.

At the command prompt, type:

```
preverify.exe [-d] output -classpath directory input; directory
```

Application distribution over the wireless network

You can distribute your applications over the wireless network to help provide a better experience to BlackBerry® device users and to simplify application distribution to a large group of people since you do not require a computer application. A BlackBerry device user can install your applications over the wireless network.

Wireless pull (user-initiated)

You can post compiled applications on a public or private web site. BlackBerry® device users can visit the web site to download the applications over the wireless network by using the browser on their BlackBerry devices. The browser prompts the users to install the application and then the application downloads over the wireless network and installs on the BlackBerry device.

Wireless push (server-initiated)

In the BlackBerry® Enterprise Server environment, the administrator can push applications to BlackBerry device users over the wireless network for mandatory installation. The administrator creates a new policy and specifies that the BlackBerry device requires the application. The application is pushed to users without any user interaction required. Organizations might find this approach useful when sending new applications to a large number of BlackBerry device users.

Distributing BlackBerry Java Applications over the wireless network

Extract sibling .cod files.

To ensure a BlackBerry® device user does not override the original .cod file, on the content server, extract the sibling .cod files into a different directory than the directory where the original file exists.

1. Unzip the original .cod file and extract the sibling .cod files.

2. Place each sibling .cod file on a content server.
3. In the .jad file, list the sibling .cod files separately. Use the following naming convention for sibling .cod files: *name of original .cod file - sequential number*.
4. Create a RIM-COD-URL - <#> parameter for each sibling .cod file, and place the name of the sibling file to the right of this parameter. # is a number that starts at 1 and increases by 1 for each sibling file. Give each sibling .cod files the same name as the original .cod file, followed by -<#>.
5. Create a RIM-COD-Size - <#> parameter for each sibling .cod file, and place the size of the sibling file to the right of this parameter. # is the same number that is appended to the name of the sibling file. Place the RIM-COD-Size - <#> parameter immediately below the RIM-COD=URL - <#> parameter.

Example: Listing sibling .cod files in a .jad file

The following example contains two sibling files named **myApp-1.cod** and **myApp-2.cod**, after the original .cod file myAPP. The developer appends the '.cod' file extension to each sibling file name. The developer creates a RIM-COD-Size - <#> parameter for each sibling file.

```
Manifest-Version: 1.0
MIDlet-Version: 1.0.0
MIDlet-1: ,,
RIM-COD-Module-Dependencies: net_rim_cldc
MicroEdition-Configuration: CLDC-1.0
RIM-COD-Module-Name: MyApp
MIDlet-Name: My Application
RIM-COD-URL: myApp.cod
RIM-COD-Size: 55000
RIM-COD-URL-1: myApp-1.cod
RIM-COD-Size-1: 50000
RIM-COD-URL-2: myApp-2.cod
RIM-COD-Size-2: 25000
MicroEdition-Profile: MIDP-1.0
```

Modifying information for a MIDlet suite

You can use the Updatejad tool, part of the BlackBerry® Java® Development Environment, to process .jad files and perform the following actions:

- Correct the .cod file sizes listed in a .jad file. The .cod file sizes listed in the .jad file change after you use the BlackBerry® Signing Authority Tool to sign .cod files.
- Create .jad files that reference multiple .cod files.

Use the Updatejad tool only on .jad files created using the BlackBerry® Integrated Development Environment or the RAPC command-line tool, and signed using the BlackBerry Signing Authority Tool .

The Updatejad tool commands have the following format:

```
updatejad.exe -q -n input.jad [additional.jad]
```

Option	Description
-q	This option suppresses the creation of success output messages for .jad file processing operations. If an error occurs during .jad file processing, a non-zero exit code is produced.
-n	This option suppresses the backup of the original .jad file.
<i>input.jad</i>	This option specifies the .jad file to update.
<i>additional.jad</i>	This option specifies other attributes to add to the input.jad file.

For more information, see the *BlackBerry Integrated Development Environment Help* or the *BlackBerry Signing Authority Tool version 1.0 - Password Based Administrator Guide*.

Correct the .cod file sizes listed in a .jad file.

1. Use the BlackBerry® Integrated Development Environment to create two BlackBerry® device application files, for example, **test.cod** and **test.jad**.
2. Use the BlackBerry® Signing Authority Tool to sign the .cod file.
3. From a command-prompt, navigate to the location of the Updatejad tool.
4. Type a command to correct the .cod file sizes listed in test.jad.

```
updatejad.exe test.jad
```

Create .jad files that reference multiple .cod files.

1. Use the BlackBerry® Integrated Development Environment to create two BlackBerry® device application files, for example, **lib.cod** and **lib.jad**.
2. Use the BlackBerry® Signing Authority Tool to sign the .cod file.
3. Use the BlackBerry IDE to create two other BlackBerry device application files that use the .jad file, for example, **test.cod** and **test.jad**.
4. Use the BlackBerry Signing Authority Tool to sign the new .cod file.
5. From a command-prompt, navigate to the location of the Updatejad tool.
6. Type a command to add the .cod file names from the first .jad file to the new one.

```
updatejad.exe test.jad lib.jad
```

Distributing BlackBerry device applications with the BlackBerry Desktop Software

Elements in BlackBerry device application .alx file

Element	Attributes	Description
application	id platformVersion blackBerryVersion	<p>The <code>application</code> element contains the elements for a single BlackBerry® device application.</p> <p>The <code>application</code> element can also contain additional nested <code>application</code> elements. Nesting lets you require that, when a BlackBerry device application is loaded on the BlackBerry device, its prerequisite modules are also loaded on the BlackBerry device.</p> <p>The <code>id</code> attribute specifies a unique identifier for the BlackBerry device application. To provide uniqueness, use an ID that includes your company domain in reverse order. For example, <code>com.rim.samples.docs.helloworld</code>.</p> <p>The <code>platformVersion</code> attribute specifies the version of the operating system software on a BlackBerry device that a BlackBerry device application requires.</p> <p>The <code>blackBerryVersion</code> attribute specifies the version of the BlackBerry® Device Software that a BlackBerry device application requires.</p>
copyright	—	The <code>copyright</code> element provides copyright information, which appears in the application loader tool of the BlackBerry® Desktop Manager.
description	—	The <code>description</code> element provides a brief description of the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager.

Element	Attributes	Description
directory	platformVersion blackBerryVersion	<p>The <code>directory</code> element provides the location of a set of files. The <code>directory</code> element is optional. If you do not specify a <code>directory</code>, the files must exist in the same location as the <code>.alx</code> file. The <code>directory</code> element specifies the directory relative to the location of the <code>.alx</code> file.</p> <p><code>directory</code> elements are cumulative within a BlackBerry device application.</p> <p>For example:</p> <pre><application id="com.abc.my.app"> <directory>MyCodFiles</directory> <fileset Java="1.0"> <files> a.cod //resolves to <.alx location> \MyCodFiles b.cod </files> </fileset> <directory>MyCodFiles</directory> <fileset Java="1.0"> <files> c.cod //resolves to <.alx location> \MyCodFiles\MyCodFiles d.cod </files> </fileset> </application></pre> <p>The <code>platformVersion</code> attribute specifies the version of the operating system software on a BlackBerry device that a BlackBerry device application requires.</p> <p>The <code>blackBerryVersion</code> attribute specifies the version of the BlackBerry Device Software that a BlackBerry device application requires.</p>
files	—	<p>The <code>files</code> element provides a list of one or more BlackBerry device application <code>.cod</code> files, in a single directory, to load on the BlackBerry device.</p>

Element	Attributes	Description
<code>fileset</code>	<p><code>Java</code></p> <p><code>radio</code></p> <p><code>langid</code></p> <p><code>Colour</code></p> <p><code>platformVersion</code></p> <p><code>blackBerryVersion</code></p>	<p>The <code>fileset</code> element includes an optional <code>directory</code> element and one or more <code>files</code> elements. It specifies a set of <code>.cod</code> files, in a single directory, to load onto the BlackBerry device. To load files from more than one directory, include one or more <code>fileset</code> elements in the <code>.alx</code> file.</p> <p>The <code>Java</code> attribute specifies the minimum version of the BlackBerry® Java® Virtual Machine with which the <code>.cod</code> files are compatible. The <code>Java</code> attribute is required.</p> <p>The <code>radio</code> attribute lets you load different BlackBerry device applications or modules on the BlackBerry device depending on the network type of the BlackBerry device. Possible values include <code>Mobitex</code>, <code>DataTAC</code>, <code>GPRS</code>, <code>CDMA</code>, and <code>IDEN</code>. The <code>radio</code> attribute is optional.</p> <p>The <code>langid</code> attribute lets you load different BlackBerry device applications or modules depending on the language support that BlackBerry device users add to the BlackBerry device. The value is a Win32 langid code; for example: <code>0x0009</code> (English), <code>0x0007</code> (German), <code>0x000a</code> (Spanish), and <code>0x000c</code> (French). The <code>langid</code> attribute is optional.</p> <p>The <code>colour</code> attribute lets you load different BlackBerry device applications or modules for color or monochrome displays. The value is a <code>Boolean</code>; <code>true</code> means color display and <code>false</code> means monochrome.</p> <p>The <code>platformVersion</code> attribute specifies the version of the operating system software on a BlackBerry device that a BlackBerry device application requires.</p> <p>The <code>blackBerryVersion</code> attribute specifies the version of the BlackBerry Device Software that a BlackBerry device application requires.</p>

Element	Attributes	Description
hidden	—	<p>The <code>hidden</code> element hides a package so that it does not appear to BlackBerry device users in the Application Loader. To hide a package, add the following line: <code><hidden>true</hidden></code>.</p> <p>Use this element in conjunction with the <code>required</code> element to load the BlackBerry device application by default, or set the <code>requires</code> tag to load this package if another BlackBerry device application exists.</p> <p>Only corporate system administrators should use the <code>hidden</code> tag. This tag is not intended for use by third-party software vendors.</p> <p>The BlackBerry Desktop Software version 3.6 or later supports this element.</p>
language	langid	<p>The <code>language</code> element lets you override the text that appears in the Application Loader when the Application Loader runs in the language that the <code>langid</code> attribute specifies.</p> <p>To support multiple languages, specify multiple <code>language</code> elements. To specify the <code>name</code>, <code>description</code>, <code>version</code>, <code>vendor</code>, and <code>copyright</code> elements for each language, nest these elements in the <code>language</code> element. If you do not nest a element, the text appears in the default language.</p> <p>The <code>langid</code> attribute specifies the Win32 <code>langid</code> code for the language to which this information applies. For example, some Win32 <code>langid</code> codes are: 0x0009 (English), 0x0007 (German), 0x000a (Spanish), and 0x000c (French).</p>
library	id	<p>You can use the <code>library</code> element instead of the <code>application</code> element. It contains the elements for a single library module. You cannot nest modules. By default, a library module does not appear in the application loader tool of the BlackBerry Desktop Manager.</p>

Element	Attributes	Description
		<p>Typically, use the library element as the target of a <code><requires></code> element, so that when a particular BlackBerry device application loads onto the BlackBerry device, a required library also loads onto the BlackBerry device.</p> <p>BlackBerry Desktop Software version 3.6 or later supports this element.</p>
loader	version	<p>The loader element contains one or more application element.</p> <p>The version attribute specifies the version of the application loader tool of the BlackBerry Desktop Manager.</p>
name	—	The name element provides a descriptive name for the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager.
required	—	<p>The required element lets you force an application to load. The application loader tool of the BlackBerry Desktop Manager selects the BlackBerry device application for installation, and the BlackBerry device user cannot change this selection. Add the following line: <code><required>true</required></code>.</p> <p>Only corporate system administrators should use the required tag. This tag should not be used by third-party software vendors.</p> <p>BlackBerry Desktop Software version 3.5 or later supports this element.</p>
requires	id	The requires element is an optional element that specifies the id of a package on which this BlackBerry device application depends. This element can appear more than once, if the BlackBerry device application depends on more than one BlackBerry device application. When a BlackBerry device application loads onto the BlackBerry device, all packages that the <code><requires></code> tag specifies also load onto the BlackBerry device.

Element	Attributes	Description
		BlackBerry Desktop Software version 3.6 or later supports this element.
vendor	—	The <code>vendor</code> element provides the name of the company that created the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager.
version	—	The <code>version</code> element provides the version number of the BlackBerry device application, which appears in the application loader tool of the BlackBerry Desktop Manager. This version number is for information purposes only.

Properties of BlackBerry device application .jad files

The BlackBerry® Integrated Development Environment lets you create a dual-purpose .jad file to support the downloading of MIDlets onto BlackBerry devices and other wireless devices. To do this, create a .jad file that contains both the `RIM-COD-URL` and `RIM-COD-Size` attributes and the `MIDlet-Jar-URL` and `MIDlet-Jar-Size` attributes. On BlackBerry devices, download the .cod files; on other wireless devices, download the .jar files.

Required RIM attribute	Description
<code>RIM-COD-Creation-Time</code>	creation time of the .cod file
<code>RIM-COD-Module-Dependencies</code>	list of modules that the .cod file requires
<code>RIM-COD-Module-Name</code>	name of the module that the .cod file contains
<code>RIM-COD-SHA1</code>	SHA1 hash of the .cod file
<code>RIM-COD-Size</code>	size (in bytes) of the .cod file
<code>RIM-COD-URL</code>	URL from which the .cod file can be loaded

Optional RIM attribute	Description
<code>RIM-Library-Flags</code>	reserved for use by Research In Motion
<code>RIM-MIDlet-Flags</code>	reserved for use by RIM
<code>RIM-MIDlet-NameResourceBundle</code>	name of the resource bundle on which the BlackBerry device application depends
<code>RIM-MIDlet-Position</code>	suggested position of the BlackBerry device application icon on the Home screen might not be the actual position of the icon on the Home screen

Application distribution through a computer connection

Distribute an application from a computer

You can use the application loader tool in the BlackBerry® Desktop Manager to install applications on the BlackBerry device. The application loader tool can provide users with a simple way to download applications from their computers to their BlackBerry devices.

Distribute an application from a web page

You can use the BlackBerry® Application Web Loader to post your compiled application on a web site. Users can use Windows® Internet Explorer® on their computers to visit the web page and install the application on their BlackBerry devices. When BlackBerry device users visit the web page, the BlackBerry Application Web Loader prompts them to connect their devices to the USB port. They can then install the application using an ActiveX® control. The BlackBerry Application Web Loader can provide BlackBerry device users with a simple way to install applications from their computers without running the BlackBerry® Desktop Manager.

Distribute an application for testing

The BlackBerry® Java® Development Environment includes a command line tool called the JavaLoader tool that is located in the BIN folder in the BlackBerry JDE folder. You can use the JavaLoader tool to quickly install and remove compiled application files on the BlackBerry device directly over the USB port. You do not require any descriptor files or web pages. The JavaLoader tool can be useful when you install and remove your application frequently during testing and development; however, the JavaLoader tool is not designed for use by BlackBerry device users.

Distributing an application from a computer

Create an application loader file

You use an application loader file to distribute a BlackBerry® device application using the Application Loader tool of the BlackBerry® Desktop Manager

1. Create an .alx file for each BlackBerry device application, and then distribute the .alx file with the .cod files to BlackBerry device users. See the *Application Loader Online Help* for more information about .alx files.
2. In the BlackBerry® Integrated Development Environment, select a project.
3. On the **Project** menu, click **Generate .alx File**.

Install a BlackBerry device application on a specific device

1. Open a text editor.
2. Locate the .alx file for the BlackBerry® device application.
3. In the .alx file, make sure the series attribute in the fileset opening tag refers to the BlackBerry device you want to install the BlackBerry device application on.

```
<fileset series="8700" Java="1.0">
```

For more information about the series attribute, see the Platform.alx file located in the simulator directory of your BlackBerry® Java® Development Environment installation directory:

Program Files\Research In Motion\BlackBerry JDE 4.6.0\simulator.

4. Make sure the files tag contains a reference to the .cod file for your BlackBerry device application.

```
<files>
My_application.cod
</files>
```

5. Update the application, description, and other tags to reflect the purpose of the .alx file.

```
<application id="Push only to 8700">
...
<description>This will push the COD only to 8700s</description>
```

Code sample: Loading a BlackBerry device application on a specific BlackBerry device

```
<loader version="1.0">
<application id="Push only to 8700">
<name>Alien</name>
<description>This will push the COD only to 8700s</description>
<version>2006.02.14.1838</version>
<vendor>RIM</vendor>
<copyright>Copyright (c) 2001-2005</copyright>
<fileset series="8700" Java="1.0">
<files>
My_application.cod
</files>
</fileset>
</application>
</loader>
```

Specifying supported versions of the BlackBerry Device Software

BlackBerry® device applications that use APIs that are only available on particular versions of the BlackBerry® Device Software should specify the supported BlackBerry device versions using the `_blackberryVersion` attribute.

BlackBerry device applications for use with a BlackBerry device running on a specific platform version should specify supported platform versions using the `_platformVersion` attribute. The `_platformVersion` attribute can be used within the directory tag, the application tag, or the fileset tag.

You can use the following rules to specify a version range for the BlackBerry Device Software or the platform version:

- Square brackets [] indicate inclusive (closed) range matching.
- Round brackets () indicate exclusive (open) range matching.
- Missing lower ranges imply 0.
- Missing upper ranges imply infinity.

For example, [4.0,) indicates any version between 4.0 and infinity.

Code sample: Preventing modules from loading on versions of the BlackBerry Device Software earlier than version 4.0.

```
<application id="application_id" _blackberryVersion="[4.0,)">
...
</application>
```

Code sample: Providing alternative modules for different versions of the BlackBerry Device Software.

```
<application id="application_id">
...
<fileset _blackberryVersion="(,4.0)">
... modules for BlackBerry device software versions earlier than 4.0
</fileset>
<fileset _blackberryVersion="[4.0,)">
... modules for BlackBerry device software versions 4.0 and later
</fileset>
</application>
```

Code sample: Preventing modules from loading on versions of the platform earlier than version 2.4.0.66.

```
<application id="application_id" _platformVersion="[2.4.0.66,)">
...
</application>
```

Code sample: Providing alternative modules for different versions of the platform.

```
<application id="application_id">
...
<fileset _platformVersion="(,2.4.0.66)">
... modules for BlackBerry OS platform versions earlier than 2.4.0.66
</fileset>
<fileset _platformVersion="[2.4.0.66,)">
... modules for BlackBerry OS platform versions 2.4.0.66 and later
</fileset>
</application>
```

Glossary

11

3GPP

Third Generation Partnership Project

AES

Advanced Encryption Standard

API

application programming interface

APN

access point name

ASCII

American National Standards Institute

.alx file

A .alx file is the application descriptor that provides information about a BlackBerry Java® Application and the location of the application's .cod files to a BlackBerry device.

BlackBerry MDS

BlackBerry® Mobile Data System

CDMA

Code Division Multiple Access

COM port

communications port

EDGE

Enhanced Data Rates for Global Evolution

EVDO

Evolution Data Optimized

GAN

generic access network

GERAN

GSM-EDGE Radio Access Network

GPRS

General Packet Radio Service

GSM

Global System for Mobile communications®

HTTP

Hypertext Transfer Protocol

HTTPS

Hypertext Transfer Protocol over Secure Sockets Layer

IPPP

Internet Protocol Proxy Protocol

JSR

Java® Specification Request

MIDP

Mobile Information Device Profile

PAP

Push Access Protocol

PIN

personal identification number

RAPC

RIM Application Program Compiler

service books

Service books determine which services are available on BlackBerry devices or BlackBerry enabled devices.

SSID

service set identifier

TCP

Transmission Control Protocol

TLS

Transport Layer Security

Triple DES

Triple Data Encryption Standard

UDP

User Datagram Protocol

UMTS

Universal Mobile Telecommunications System

UTRAN

UMTS Terrestrial Radio Access Network

WAP

Wireless Application Protocol

WLAN

wireless local area network

Provide feedback

12

To provide feedback on this deliverable, visit www.blackberry.com/docsfeedback.

Legal notice

13

©2009 Research In Motion Limited. All rights reserved. BlackBerry®, RIM®, Research In Motion®, SureType®, SurePress™ and related trademarks, names, and logos are the property of Research In Motion Limited and are registered and/or used in the U.S. and countries around the world.

802.11a, 802.11b, and 802.11g are trademarks of the Institute of Electrical and Electronics Engineers, Inc. Bluetooth is a trademark of Bluetooth SIG. Global System for Mobile communications is a trademark of the GSM MOU Association. iDEN is a trademark of Motorola, Inc. Microsoft, ActiveX, Internet Explorer, and Windows are trademarks of Microsoft Corporation. Java is a trademark of Sun Microsystems, Inc. UMTS is a trademark of European Telecommunications Standard Institute. Wi-Fi is a trademark of the Wi-Fi Alliance. All other trademarks are the property of their respective owners.

The BlackBerry smartphone and other devices 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; D445,428; D433,460; D416,256. Other patents are registered or pending in the U.S. and in various countries around the world. Visit www.rim.com/patents for a list of RIM (as hereinafter defined) patents.

This documentation including all documentation incorporated by reference herein such as documentation provided or made available at www.blackberry.com/go/docs is provided or made accessible "AS IS" and "AS AVAILABLE" and without condition, endorsement, guarantee, representation, or warranty of any kind by Research In Motion Limited and its affiliated companies ("RIM") and RIM assumes no responsibility for any typographical, technical, or other inaccuracies, errors, or omissions in this documentation. In order to protect RIM proprietary and confidential information and/or trade secrets, this documentation may describe some aspects of RIM technology in generalized terms. RIM reserves the right to periodically change information that is contained in this documentation; however, RIM makes no commitment to provide any such changes, updates, enhancements, or other additions to this documentation to you in a timely manner or at all.

This documentation might contain references to third-party sources of information, hardware or software, products or services including components and content such as content protected by copyright and/or third-party web sites (collectively the "Third Party Products and Services"). RIM does not control, and is not responsible for, any Third Party Products and Services including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third Party Products and Services. The inclusion of a reference to Third Party Products and Services in this documentation does not imply endorsement by RIM of the Third Party Products and Services or the third party in any way.

EXCEPT TO THE EXTENT SPECIFICALLY PROHIBITED BY APPLICABLE LAW IN YOUR JURISDICTION, ALL CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS, OR WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS OR WARRANTIES OF DURABILITY, FITNESS FOR A PARTICULAR PURPOSE OR USE, MERCHANTABILITY, MERCHANTABLE QUALITY, NON-INFRINGEMENT, SATISFACTORY QUALITY, OR TITLE, OR ARISING FROM A STATUTE OR CUSTOM OR A COURSE OF DEALING OR USAGE OF TRADE, OR RELATED TO THE DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN, ARE HEREBY EXCLUDED. YOU MAY ALSO HAVE OTHER RIGHTS THAT VARY BY STATE OR PROVINCE. SOME JURISDICTIONS MAY NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES AND CONDITIONS. TO THE EXTENT

PERMITTED BY LAW, ANY IMPLIED WARRANTIES OR CONDITIONS RELATING TO THE DOCUMENTATION TO THE EXTENT THEY CANNOT BE EXCLUDED AS SET OUT ABOVE, BUT CAN BE LIMITED, ARE HEREBY LIMITED TO NINETY (90) DAYS FROM THE DATE YOU FIRST ACQUIRED THE DOCUMENTATION OR THE ITEM THAT IS THE SUBJECT OF THE CLAIM.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, IN NO EVENT SHALL RIM BE LIABLE FOR ANY TYPE OF DAMAGES RELATED TO THIS DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN INCLUDING WITHOUT LIMITATION ANY OF THE FOLLOWING DAMAGES: DIRECT, CONSEQUENTIAL, EXEMPLARY, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR AGGRAVATED DAMAGES, DAMAGES FOR LOSS OF PROFITS OR REVENUES, FAILURE TO REALIZE ANY EXPECTED SAVINGS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF BUSINESS OPPORTUNITY, OR CORRUPTION OR LOSS OF DATA, FAILURES TO TRANSMIT OR RECEIVE ANY DATA, PROBLEMS ASSOCIATED WITH ANY APPLICATIONS USED IN CONJUNCTION WITH RIM PRODUCTS OR SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF RIM PRODUCTS OR SERVICES OR ANY PORTION THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR SERVICES, COST OF CAPITAL, OR OTHER SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES WERE FORESEEN OR UNFORESEEN, AND EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, RIM SHALL HAVE NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO RIM AND ITS AFFILIATED COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE PROVIDERS), AUTHORIZED RIM DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND THEIR RESPECTIVE DIRECTORS, EMPLOYEES, AND INDEPENDENT CONTRACTORS.

IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF RIM OR ANY AFFILIATES OF RIM HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with RIM's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for acquiring them. You should not install or use Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and Services that are provided with RIM's products and services are provided as a convenience to you and are provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations, or warranties of any kind by RIM and RIM assumes no liability whatsoever, in relation thereto. Your use of Third Party Products and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other agreements applicable thereto with third parties, except to the extent expressly covered by a license or other agreement with RIM.

Certain features outlined in this documentation require a minimum version of BlackBerry® Enterprise Server, BlackBerry® Desktop Software, and/or BlackBerry® Device Software.

The terms of use of any RIM product or service are set out in a separate license or other agreement with RIM applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY RIM FOR PORTIONS OF ANY RIM PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.

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