

BlackBerry Java Application

MIDlet

Version: 5.0

Development Guide

Contents

1	Java ME and Java APIs for BlackBerry.....	3
2	Advantages of MIDlets.....	4
3	Disadvantages of MIDlets.....	5
4	Creating a basic MIDlet.....	6
5	Enhancing a basic MIDlet.....	7
	Displaying items on the screen.....	7
	Display an image.....	7
	Code sample: Displaying an image.....	10
	Drawing on the screen.....	11
	Draw a rectangle.....	11
	Code sample: Drawing a rectangle.....	12
	Draw a circle.....	13
	Code sample: Drawing a circle.....	14
	Displaying controls.....	15
	Display a label.....	16
	Code sample: Displaying a sample.....	17
	Display a text field.....	18
	Code sample: Displaying a text field.....	20
	Display option buttons.....	20
	Code sample: Displaying option buttons.....	23
	Display check boxes.....	23
	Code sample: Displaying check boxes.....	25
	Display commands.....	26
	Code sample: Displaying commands.....	28
	Display a drop-down.....	29
	Code sample: Displaying a drop-down.....	31
	Handling input.....	31
	Respond to commands.....	32
	Code sample: Respond to commands.....	34
	Responding to control input.....	35

6	Glossary.....	57
7	Provide feedback.....	58
8	Document revision history.....	59
9	Legal notice.....	60

Java ME and Java APIs for BlackBerry

1

Java® ME is an industry standard platform that defines common sets of Java APIs for different types of wireless and embedded devices. A Java ME application on a BlackBerry® device runs in the BlackBerry® Java® Virtual Machine, which provides all of the runtime services to the applications and performs functions such as typical memory allocations, security checks, and garbage collection.

The Java ME MIDP standard addresses the API and BlackBerry JVM needs of a constrained wireless device with a user interface. The BlackBerry device supports the Java ME MIDP standard as defined in JSR 118. BlackBerry devices that run BlackBerry Device Software version 5.0 or later support MIDP 2.1. The Java ME MIDP standard provides a core set of Java APIs that any BlackBerry device can support, regardless of its underlying operating system. Developers can often build one Java application using the MIDP standard APIs and run that application on many different types of devices.

Advantages of MIDlets

2

Although the BlackBerry® Java® APIs give you access to more features than the MIDP API, MIDlets have some advantages over BlackBerry Java applications.

Advantage	Description
Cross platform	MIDlets run on any Java based device that supports the version of CLDC and MIDP that the MIDlet uses.
Availability of learning resources	Because MIDlets can run on various devices, many vendors publish information about MIDlet development. Many MIDlet code examples are publicly available, as are a number of books, articles, and websites about MIDlet development.
Thread safe UI APIs	Except for the <code>ServiceRepaints()</code> method of the <code>Canvas</code> class, you can use the UI APIs without concern for threading issues.

Disadvantages of MIDlets

3

The advantages MIDlets have over BlackBerry® Java® applications come at the cost of a number of disadvantages.

Disadvantage	Description
Limited feature access	Many of the features of BlackBerry devices are made available through the BlackBerry Java API. Only a subset of those features can be leveraged using the MIDlet API.
Poor screen layout functionality	The MIDlet API provides only the simplest mechanisms for easily positioning UI elements on the device screen. The result is that, even for relatively simple layouts, you have to resort to writing code to draw directly to the screen.

Creating a basic MIDlet

4

MIDlets are wireless applications that run in a highly-controlled runtime environment. Application management software manages the lifetime of a MIDlet, causing it to transition between three states: active, paused and destroyed. The `MIDlet` class represents the application and specifies the interface to implement to enable the application management software to control the MIDlet.

To create the most basic MIDlet, you must extend the `MIDlet` class and provide implementations for the three abstract methods of the class: `startApp()`, `pauseApp()` and `destroyApp()`. The `MIDlet` class is implemented in the `java.microedition.midlet` library, so you must import that library to create a MIDlet.

The following listing illustrates the code required to create the most basic MIDlet.

```
import javax.microedition.midlet.*;

public class MostBasicMIDlet extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}
```

To learn more about the `MIDlet` class, see the API reference for BlackBerry® Java® Development Environment.

Enhancing a basic MIDlet

5

After you successfully compile and install a basic MIDlet, you have a firm foundation upon which to build. To enhance a MIDlet, add a small amount of new functionality, such as a new UI control for example. After you add the functionality, compile and test the MIDlet before you enhance it further.

Displaying items on the screen

To display items on a BlackBerry® device screen you can use prebuilt UI classes or create your own UI classes. Prebuilt UI classes model the screens and the items you can include on screens. You can quickly create a UI using prebuilt UI classes, but they are not very flexible. The other approach is to create your own UI classes.

To create your own UI classes, you can extend the `CustomItem` class or draw directly on the screen by overriding the `paint()` method of the `Canvas` class. The approach that uses the `Canvas` class is very flexible, but requires significant effort. MIDP-compliant devices have widely-varying screen sizes. To draw UI elements you must programmatically retrieve the screen metrics and use the metrics to properly position and size the UI elements.

Display an image

To display an image on the screen of a BlackBerry® device, create an `Image` object and populate it by calling the static `Image.createImage()` method. Provide the location of the image as a parameter.

You must add the image to the project. In the BlackBerry® Java® Development Environment, check that it shows up along with the Java® source files in the tree control within the BlackBerry® Integrated Development Environment.

1. Import the three required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`.

```
public class DisplayAnImage extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }
}
```

```

    public void destroyApp(boolean flag)
    {
        }
}

```

3. Create private variables to store instances of the `Display` and `Form` classes. Also create a private variable in which to store the image.

```

public class DisplayAnImage extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private Image pngBackground;

    // MIDlet lifecycle method overrides omitted
}

```

4. In the overridden version of the `startApp()` method, create an instance of the `ImageItem` object, populate it with an image and add it to a `Form` object.
 - a. Use the static `createImage()` method of the `Image` class to create an instance of the `Image` class that represents the image you want to display.

```

public void startApp()
{
    pngBackground = Image.createImage("test_image.png");
}

```

- b. Create a new instance of the `ImageItem` class. The constructor takes as parameters a label, an `Image` object, a layout, alternative text and an appearance mode.

```

public void startApp()
{
    pngBackground = Image.createImage("test_image.png");
    ImageItem img = new ImageItem("bg", pngBackground,
    ImageItem.LAYOUT_EXPAND, "background", ImageItem.PLAIN);
}

```

- c. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add the image to the form.

```

public void startApp()
{
    pngBackground = Image.createImage("test_image.png");
    ImageItem img = new ImageItem("bg", pngBackground,
    ImageItem.LAYOUT_EXPAND, "background", ImageItem.PLAIN);
}

```

```

    mForm = new Form("MIDlet Developer Guide: Display an image.");
    mForm.append(img);
}

```

- d. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable object to the form stored in the `mForm` variable.

```

public void startApp()
{
    pngBackground = Image.createImage("test_image.png");
    ImageItem img = new ImageItem("bg",pngBackground,
    ImageItem.LAYOUT_EXPAND,"background",ImageItem.PLAIN);

    mForm = new Form("MIDlet Developer Guide: Display an image.");
    mForm.append(img);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}

```

- e. Add a try-catch block in case the image file cannot be accessed.

```

public void startApp()
{
    try
    {
        pngBackground = Image.createImage("test_image.png");
        ImageItem img = new ImageItem("bg",pngBackground,
        ImageItem.LAYOUT_EXPAND,"background");

        mForm = new Form("MIDlet Developer Guide: Display an image.");
        mForm.append(img);

        mDisplay = Display.getDisplay(this);
        mDisplay.setCurrent(mForm);

    }
    catch(IOException e)
    {
        mForm.append(e.getMessage());
    }
}

```

Code sample: Displaying an image

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class DisplayAnImage extends MIDlet
{
    private Form mForm;
    private Display mDisplay;
    private Image pngBackground;

    public DisplayAnImage()
    {
    }

    public void destroyApp(boolean flag)
    {
    }

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mForm = new Form("MIDlet Developer Guide: Display an image.");

        try
        {
            pngBackground = Image.createImage("test_image.png");
            ImageItem img = new ImageItem("bg",pngBackground,
            ImageItem.LAYOUT_EXPAND,"background");
            mForm.append(img);
            mDisplay.setCurrent(mForm);
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }
    }

    public void pauseApp()
    {
    }
}
```

Drawing on the screen

To draw on the screen you must extend the `Canvas` class and override the `paint()` method. To redraw the screen, the MIDlet application environment calls your `paint()` method and passes the method a `Graphics` object. The object provides access to information about the screen and enables you to draw simple shapes and present images and text.

Code sample: Overriding the `paint()` method to draw a colored circle

In the following code sample, the `getWidth()` and `getHeight()` methods are used to retrieve the dimensions of the screen on the BlackBerry® device that is running the code. A circle is then positioned near the center of the screen by specifying half the width and half the height as the first two parameter values in `drawArc()` and `fillArc()`.

```
class ShapeCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawArc(width/2,height/2,50,50,0,360);
        g.fillArc(width/2,height/2,50,50,0,360);
    }
}
```

Draw a rectangle

To draw a rectangle on the BlackBerry® device screen, you must extend the `Canvas` class and override its `paint()` method.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. To create the framework for the MIDlet, extend the `MIDlet` class and override the mandatory methods: `startApp()`, `pauseApp()`, and `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }
}
```

```

    public void destroyApp(boolean flag)
    {
    }
}

```

3. Extend the `Canvas` class and override the `paint()` method, using `drawRect()` and `fillRect()` to draw the outline of a rectangle and fill it in.

```

class RectangleCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawRect(width/2,height/2,70,30);
        g.fillRect(width/2,height/2,70,30);
    }
}

```

4. In the overridden version of the `startApp()` method, create an instance of the custom `RectangleCanvas` class. Use the static `setCurrent()` method of the `Display` object to set the current `Displayable` to the instance of `RectangleCanvas`.

```

public void startApp()
{
    Displayable disp = new RectangleCanvas();
    Display.getDisplay(this).setCurrent(disp);
}

```

Code sample: Drawing a rectangle

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DrawARectangle extends MIDlet
{
    public void startApp()
    {
        Displayable disp = new RectangleCanvas();
        Display.getDisplay(this).setCurrent(disp);
    }

    public void destroyApp(boolean flag)
    {

```

```

    }

    public void pauseApp()
    {
    }
}

class RectangleCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawRect(width/2,height/2,70,30);
        g.fillRect(width/2,height/2,70,30);
    }
}

```

Draw a circle

To draw a circle on the BlackBerry® device screen, you must extend the Canvas class and override its `paint()` method.

1. Import the two required MIDlet libraries.

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

2. To create the framework for the MIDlet, extend the MIDlet class and override the three mandatory methods: `startApp()`, `pauseApp()`, and `destroyApp()`.

```

public class DrawACircle extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}

```

3. Extend the Canvas class and override the `paint()` method, using `drawArc()` and `fillArc()` to draw the outline of a circle and fill it in.

```

class CircleCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawArc(width/2,height/2,50,50,0,360);
        g.fillArc(width/2,height/2,50,50,0,360);
    }
}

```

4. In the overridden version of the `startApp()` method, create an instance of the custom `CircleCanvas` displayable. Call the static `setCurrent()` method of the `Display` object to configure the current displayable to be the instance of `CircleCanvas`.

```

public void startApp()
{
    Displayable disp = new CircleCanvas();
    Display.getDisplay(this).setCurrent(disp);
}

```

Code sample: Drawing a circle

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DrawACircle extends MIDlet
{
    public void startApp()
    {
        Displayable disp = new CircleCanvas();
        Display.getDisplay(this).setCurrent(disp);
    }

    public void destroyApp(boolean flag)
    {
    }

    public void pauseApp()
    {
    }
}

class CircleCanvas extends Canvas
{

```

```

public void paint(Graphics g)
{
    int width = getWidth();
    int height = getHeight();
    g.setColor(0xCC0999);
    g.drawArc(width/2,height/2,50,50,0,360);
    g.fillArc(width/2,height/2,50,50,0,360);
}
}

```

Displaying controls

Subclasses of the `Item` class represent controls. There are pre-built subclasses of `Item` that represent common controls. The following table lists these pre-built controls along with the UI controls they can be used to represent.

Item subclass	Represented UI controls
<code>ChoiceGroup</code>	Option button, check box and drop-down list
<code>DateField</code>	Calendar and time control
<code>Gauge</code>	<code>DateField</code>
<code>ImageItem</code>	Image control
<code>Spacer</code>	Spacing control
<code>StringItem</code>	Label
<code>TextField</code>	Edit box control

In addition to these pre-built controls, there is an abstract class called `CustomItem` that enables you to create custom controls. To create a custom control, you create a class that extends `CustomItem` and overrides the five abstract methods in that class. Four of the abstract methods are related to sizing of the control. The fifth method is `paint()`. It is in your override of the `paint()` method that you write the code to draw your custom control.

Regardless of whether you use a pre-built or custom control, the steps to display the control are the same.

- Create an instance of the control.
- Set the control properties.
- Create an instance of the `Form` class.
- Use the `append()` method of the `Form` instance to add the control to the form.
- Use the `setCurrent()` method of the `Display` class to display the form containing the control.

Display a label

To display a label on the screen of a BlackBerry® device, you have to create instances of a `Form` object and a `StringItem` object. You can pass the text to appear in the control as parameters to the class constructor.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`.

```
public class DisplayAnEditBox extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}
```

3. Create private variables to store instances of the `Display`, `Form` and `StringItem` classes.

```
public class DisplayALabel extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private StringItem mLabel;

    // MIDlet lifecycle method overrides omitted
}
```

4. In the overridden version of the `startApp()` method, create an instance of the `ChoiceGroup` object, populate it with check box choices and add it to a `Form` object.
 - a. Create an instance of the `StringItem` class to represent the label. Specify `StringItem.PLAIN` in the constructor to indicate that the label should appear as plain text. Other available appearance options cause the label to appear as a button or a hyperlink.

```
public void startApp()
{
    mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
}
```

- b. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add the label to the form.

```
public void startApp()
{
    mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
    mForm = new Form("Display a Label");
    mForm.append(mLabel);
}
```

- c. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```
public void startApp()
{
    mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
    mForm = new Form("Display a Label");
    mForm.append(mLabel);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}
```

Code sample: Displaying a sample

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DisplayALabel extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private StringItem mLabel;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
    }
}
```

```

        mForm = new Form("Display a Label");
        mForm.append(mLabel);
        mDisplay.setCurrent(mForm);
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}

```

Display a text field

To display a text field on the screen of a BlackBerry® device, you have to create instances of a `Form` object and a `TextField` object. You append the `TextField` object to the `Form` object.

1. Import the two required MIDlet libraries.

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`.

```

public class DisplayATextField extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}

```

3. Create private variables to store instances of the `Display`, `Form` and `TextField` objects. Also, create a constant called `MAXCHARS`. The constant will be used to specify the maximum number of characters that can be entered in the text field.

```

public class DisplayATextField extends MIDlet
{
    private Display mDisplay;

```

```

private Form mForm;
private TextField mTextField;
private static final int MAXCHARS = 100;

// MIDlet lifecycle method overrides omitted
}

```

4. In the overridden version of the `startApp()` method, create an instance of the `TextField` object and add it to a `Form` object.
 - a. Create an instance of the `TextField` class to represent the text field. Specify `MAXCHARS` in the constructor to limit the number of characters that can be input into the text field.

```

public void startApp()
{
    mEditBox = new TextField("Text Field Label:",null,MAXCHARS,0);
}

```

- b. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add the text field to the form.

```

public void startApp()
{
    mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
    mForm = new Form("MIDlet Developer Guide: Display a text field.");
    mForm.append(mTextField);
}

```

- c. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable object to the form stored in the `mForm` variable.

```

public void startApp()
{
    mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
    mForm = new Form("MIDlet Developer Guide: Display a text field.");
    mForm.append(mEditBox);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}

```

Code sample: Displaying a text field

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DisplayATextField extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private TextField mTextField;
    private static final int MAXCHARS = 100;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
        mForm = new Form("MIDlet Developer Guide: Display a text field.");
        mForm.append(mTextField);
        mDisplay.setCurrent(mForm);
    }

    public void destroyApp(boolean flag)
    {
    }

    public void pauseApp()
    {
    }
}
```

Display option buttons

To display option buttons on the screen of a BlackBerry® device, you have to create instances of a `Form` object and a `ChoiceGroup` object. You append the option button choices you want to the `ChoiceGroup` object and then append the `ChoiceGroup` object to the `Form` object.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
```

```

public void startApp()
{
}

public void pauseApp()
{
}

public void destroyApp(boolean flag)
{
}
}

```

3. Create private variables to store instances of the `Display`, `Form` and `ChoiceGroup` objects.

```

public class DisplayOptionButtons extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;

    // MIDlet lifecycle method overrides omitted
}

```

4. Create private variables to store instances of the `Display`, `Form` and `ChoiceGroup` objects.

```

public class DisplayOptionButtons extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;

    // MIDlet lifecycle method overrides omitted
}

```

5. Create an instance of the `ChoiceGroup` class to represent the set of option buttons. Specify `Choice.EXCLUSIVE` in the `ChoiceGroup` constructor to indicate that the group of choices are mutually exclusive and should, therefore, be presented as option buttons. Use the `append()` method of the `ChoiceGroup` instance to create and add a label to each option button.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
}

```

6. In the overridden version of the `startApp()` method, create an instance of the `ChoiceGroup` object, populate it with option button choices and add it to a `Form` object.
- Create an instance of the `ChoiceGroup` class to represent the set of option buttons. Specify `Choice.Exclusive` in the `ChoiceGroup` constructor to indicate that the group of choices are mutually exclusive and should, therefore, be presented as option buttons. Use the `append()` method of the `ChoiceGroup` instance to create and add a label to each option button choice.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
}
```

- Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add to the form the set of option buttons stored in the `mChoices` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Display Option Buttons");
    mForm.append(mChoices);
}
```

- Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Display Option Buttons");
    mForm.append(mChoices);
}
```

```
mDisplay = Display.getDisplay(this);
mDisplay.setCurrent(mForm);
}
```

Code sample: Displaying option buttons

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DisplayOptionButtons extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mForm = new Form("Display Option Buttons");

        mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);

        mForm.append(mChoices);
        mDisplay.setCurrent(mForm);
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}
```

Display check boxes

To display check boxes on the screen of a BlackBerry® device, you must create a `Form` object and a `ChoiceGroup` object. You must append the check box choices to the `ChoiceGroup` object and then append the `ChoiceGroup` object to the `Form` object.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. To create the framework for the MIDlet, extend the MIDlet class and override the mandatory methods: `startApp()`, `pauseApp()`, and `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}
```

3. Create private variables to store instances of the `Display`, `Form` and `ChoiceGroup` objects.

```
public class DisplayCheckBoxes extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;

    // MIDlet lifecycle method overrides omitted
}
```

4. In the overridden version of the `startApp()` method, create an instance of the `ChoiceGroup` object, populate it with checkbox choices and add it to a `Form` object.
 - a. Create an instance of the `ChoiceGroup` class to represent the set of checkboxes. Specify `Choice.MULTIPLE` in the `ChoiceGroup` constructor to indicate that the you can select more than one of the presented choices and they should, therefore, be presented as check boxes. Use the `append()` method of the `ChoiceGroup` instance to create and add a label to each check box choice.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display check
boxes",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
}
```

- b. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add to the form the set of check boxes stored in the `mChoices` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display check
boxes",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Display Check Boxes");

    mForm.append(mChoices);
}
```

- c. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display check
boxes",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Display Check Boxes");
    mForm.append(mChoices);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}
```

Code sample: Displaying check boxes

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DisplayCheckBoxes extends MIDlet
{
    Display mDisplay;
    Form mForm;
    ChoiceGroup mChoices;
```

```

public void startApp()
{
    mDisplay = Display.getDisplay(this);
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display
checkboxes",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Display Check Boxes");
    mForm.append(mChoices);
    mDisplay.setCurrent(mForm);
}

public void destroyApp(boolean flag)
{
}

public void pauseApp()
{
}
}

```

Display commands

To create a user interface using the MIDP API, you first retrieve an instance of the `Display` class that represents the screen on the BlackBerry® device. You next create an instance of a class that represents something that can be displayed on the screen of a device. All such classes are derived from the `Displayable` class. Finally, you use the `setCurrent()` method of the `Display` class, passing it the instance of the `Displayable` class, to display something on the device screen.

All classes that derive from `Displayable` inherit the ability to have commands associated with them. A command is a user interface element that a MIDP-compliant device can render in a number of different ways. It might be displayed as a button, a menu or in some other way that enables the device user to interact with it. When you write code to create a command, you cannot control exactly how the command will appear or where it will appear. You can provide both a short and long textual label for the command and you can specify the type of the command and assign it a relative priority. In the code that describes the command, you declare what you would like and each device is free to interpret those declarations and implement them in a way that makes the most sense for that particular device.

Commands are represented by the `Command` class. To display a command, you create a new instance of the `Command` class and associate it with an instance of a `Displayable`. You establish the association by passing the instance of the `Command` class as a parameter to the `addCommand()` method of the `Displayable` class.

1. Import the two required MIDP libraries. The `MIDlet` class is in the `midlet` namespace and the `Command` and `Form` classes are in the `lcdui` namespace.

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

2. Create the framework for the MIDlet by extending the MIDlet class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`.

```
public class DisplayCommands extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}
```

3. Add private member variables. Add private member variables to your derived MIDlet application class to store references to instances of the `Display`, `Form` and `Command` classes.

```
public class DisplayCommands extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private Command mCommand;

    //MIDlet lifecycle methods omitted
}
```

4. Populate the private member variables. In the overridden version of the `startApp()` method, use the static `getDisplay()` method of the `Display` class to retrieve an instance of the class corresponding to the display on the current device. Populate the `mDisplay` variable with the returned instance. Create new instances of the `Form` and `Command` classes and store references to them in the `mForm` and `mCommand` member variables. The string passed to the `Form` class constructor will appear as a title on the form when it is displayed on the device. The first two strings passed to the `Command` class constructor are short and long labels. The third parameter indicates the type of the command. In this case, `Command.SCREEN` is specified to indicate that the command is not a standard command like `Back` or `Help`. The final parameter sets the relative priority of the command. There is only one command in this example, but the value `0` is specified to ensure that this command is treated with high relative priority if other commands are added in the future.

```
public void startApp()
{
    mDisplay = Display.getDisplay(this);
    mForm = new Form("MIDlet Developers Guide: Display Commands");
    mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
}
```

5. Associate the command with the `Displayable`.

The `Displayable` is a form in this example. The `addCommand()` method of the `Form` class is used to associate the command instance stored in `mCommand` to the form referenced by `mForm`.

```
public void startApp()
{
    mDisplay = Display.getDisplay(this);
    mForm    = new Form("MIDlet Developers Guide: Display Commands");
    mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
    mForm.addCommand(mCommand);
}
```

6. Set the current `Displayable` to the one with the associated command.

The form referenced by `mForm` has an associated command. Use the `setCurrent()` method of the `Display` class to set the current `Displayable` to `mForm`.

```
public void startApp()
{
    mDisplay = Display.getDisplay(this);
    mForm    = new Form("MIDlet Developers Guide: Display Commands");
    mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
    mForm.addCommand(mCommand);
    mDisplay.setCurrent(mForm);
}
```

Code sample: Displaying commands

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DisplayCommands extends MIDlet
{
    Display mDisplay;
    Form    mForm;
    Command mCommand;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mForm    = new Form("MIDlet Developers Guide: Display Commands");
        mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
        mForm.addCommand(mCommand);
        mDisplay.setCurrent(mForm);
    }

    public void pauseApp()
```

```

{
}

public void destroyApp(boolean bForce)
{
}
}

```

Display a drop-down

To display a drop-down menu on the screen of a BlackBerry® device, you have to create instances of a `Form` object and a `ChoiceGroup` object. When you create the `ChoiceGroup` object, you specify that it is of type `Choice.POPUP`. This option is rendered on the device as a drop-down menu. You append the drop-down choices you want to the `ChoiceGroup` object and then append the `ChoiceGroup` object to the `Form` object.

1. Import the two required MIDlet libraries.

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`.

```

public class DrawACircle extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}

```

3. Create private variables to store instances of the `Display`, `Form` and `ChoiceGroup` objects.

```

public class DisplayCheckBoxes extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;

    // MIDlet lifecycle method overrides omitted
}

```

4. In the overridden version of the `startApp()` method, create an instance of the `ChoiceGroup` object, populate it with checkbox choices and add it to a `Form` object.
 - a. Create an instance of the `ChoiceGroup` class to represent the set of checkboxes. Specify `Choice.POPUP` in the `ChoiceGroup` constructor to indicate that the group of choices are mutually exclusive and should be presented in the form of a drop-down menu. Use the `append()` method of the `ChoiceGroup` instance to create and add a label to each drop-down choice.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
}
```

- b. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add to the form the drop-down stored in the `mChoices` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Display a drop-down");

    mForm.append(mChoices);
}
```

- c. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Display a drop-down");
    mForm.append(mChoices);
}
```

```
mDisplay = Display.getDisplay(this);
mDisplay.setCurrent(mForm);
}
```

Code sample: Displaying a drop-down

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DisplayADropDown extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down", Choice.POPUP);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);

        mForm = new Form("Display a drop-down");
        mForm.append(mChoices);
        mDisplay.setCurrent(mForm);
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}
```

Handling input

To retrieve and process input on a device, you implement a listener interface in a class used by a MIDlet. You then associate the listener with a UI class derived from `Displayable`. The UI class is notified of input and passes information about the input to the appropriate, registered listener methods for processing.

There are two listener interfaces: `CommandListener` and `ItemStateListener`. Each includes a single method that is called to process input. The single method of the `CommandListener` interface is called `commandAction()` and accepts a `Command` and a `Displayable` as parameters. If the input was initiated by a command, the value of the `Command` parameter identifies the instance of the corresponding `Command` class. The value of the `Displayable` parameter corresponds to the instance of the `Displayable` class that represents the UI component in which the event was received.

Respond to commands

To respond to commands you have to implement the `CommandListener` interface. You must also associate the listener with the form that has the commands associated with it. You make the association by using the `setCommandListener()` method of the `Form` object.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`. Implement the `CommandListener` interface by including the `commandAction()` method.

```
public class RespondToCommands extends MIDlet implements CommandListener
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }

    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

3. Create private variables to store instances of the `Display`, `Form`, and `Command` classes.

```
public class RespondToCommands extends MIDlet implements CommandListener
{
    private Display mDisplay;
```

```
private Form    mForm;
private Command mCommand1;
private Command mCommand2;
```

4. In the overridden version of the `startApp()` method, create instances of the `Command` object and add them to a `Form` object.
 - a. Create two new instances of the `Command` class to represent two distinct commands to process.

```
public void startApp()
{
    mCommand1 = new Command("Short Label 1", "A Long Command Label
1", Command.SCREEN, 0);
    mCommand2 = new Command("Short Label 2", "A Long Command Label
2", Command.SCREEN, 0);
}
```

- b. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `addCommand()` method of the `Form` object to add the commands to the form. Finally, use the `setCommandListener()` method of the `Form` class to associate a listener with the form. Passing the `this` keyword to `setCommandListener()` specifies that the listener interface is implemented by the `MIDlet` class itself. The implementation of the one class in the interface, `commandAction()` is described below.

```
public void startApp()
{
    mCommand1 = new Command("Short Label 1", "A Long Command Label
1", Command.SCREEN, 0);
    mCommand2 = new Command("Short Label 2", "A Long Command Label
2", Command.SCREEN, 0);

    mForm    = new Form("MIDlet Developerment Guide: Display Commands");
    mForm.addCommand(mCommand1);
    mForm.addCommand(mCommand2);
    mForm.setCommandListener(this);
}
```

- c. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```
public void startApp()
{
    mCommand1 = new Command("Short Label 1", "A Long Command Label
1", Command.SCREEN, 0);
    mCommand2 = new Command("Short Label 2", "A Long Command Label
2", Command.SCREEN, 0);

    mForm    = new Form("MIDlet Developers Guide: Display Commands");
    mForm.addCommand(mCommand1);
```

```

mForm.addCommand(mCommand2);
mForm.setCommandListener(this);

mDisplay = Display.getDisplay(this);
mDisplay.setCurrent(mForm);
}

```

5. To determine the command that was invoked, use an if/else construct to compare the instance of the `Command` class passed to the `commandAction()` method with the implemented commands. Display a message indicating which command was invoked by using the `append()` method of the `Form` class.

```

public void commandAction(Command c, Displayable d)
{
    if (c == mCommand1)
    {
        mForm.append("You invoked command 1.");
    }
    else if (c == mCommand2)
    {
        mForm.append("You invoked command 2.");
    }
}

```

Code sample: Respond to commands

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class RespondToCommands extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private Command mCommand1;
    private Command mCommand2;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mForm = new Form("MIDlet Developers Guide: Display Commands");
        mCommand1 = new Command("Short Label 1", "A Long Command Label 1", Command.SCREEN,
0);
        mCommand2 = new Command("Short Label 2", "A Long Command Label 2", Command.SCREEN,
0);
        mForm.addCommand(mCommand1);
        mForm.addCommand(mCommand2);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }
}

```

```

public void pauseApp()
{
}

public void destroyApp(boolean bForce)
{
}

public void commandAction(Command c, Displayable d)
{
    if (c == mCommand1)
    {
        mForm.append("You invoked command 1.");
    }
    else if (c == mCommand2)
    {
        mForm.append("You invoked command 2.");
    }
}
}

```

Responding to control input

To respond to control input, you extend the `MIDlet` class and implement the `CommandListener` interface. You associate the listener with the form that contains the controls. The form forwards input from those controls to your implementation of the `commandAction()` method of the `CommandListener` interface.

Respond to information entered in a text field

To respond to information entered in a text field you have to implement the `CommandListener` interface. You must also associate the listener with the form that contains the instance of the `TextField` class that represents the edit box. You make the association by using the `setCommandListener()` method of the `Form` object.

1. Import the two required MIDlet libraries.

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()`, and `destroyApp()`. Implement the `CommandListener` interface by including the `commandAction()` method.

```

public class RespondToTextField extends MIDlet implements CommandListener
{
    public void startApp()
    {

```

```

    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }

    public void commandAction(Command c, Displayable disp)
    {
    }
}

```

3. Create private variables to store instances of the `Display`, `Form`, `TextField` and `Command` classes. Also, create a constant called `MAXCHARS`. The constant is used to specify the maximum number of characters that can be entered in the edit box.

```

public class RespondToTextField extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private TextField mEditBox;
    private Command mSubmitCmd;
    private static final int MAXCHARS = 100;

    // MIDlet lifecycle method overrides omitted
}

```

4. In the overridden version of the `startApp()` method, create an instance of the `TextField` object and add it to a `Form` object.
 - a. Create an instance of the `TextField` class to represent the edit box. Specify `MAXCHARS` in the constructor to limit the number of characters that can be input into the edit box.

```

public void startApp()
{
    mTextField = new TextField("Text Field Label:", null, MAXCHARS, 0);
}

```

- b. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add the edit box to the form.

```

public void startApp()
{
    mTextField = new TextField("Text Field Label:", null, MAXCHARS, 0);
    mForm = new Form("MIDlet Developer Guide: Respond to information entered in

```

```

an edit box.");
    mForm.append(mEditBox);
}

```

- c. Create a new instance of the `Command` class and store it in the private `mSubmitCmd` variable. Specify `Command.OK` as the second parameter in the constructor to indicate to the MIDP framework that the command should behave like an OK or submit button. Use the `addCommand()` method of the `Form` class to associate the command with `mForm`, the instance of the `Form` class that contains the text field. Finally, use the `setCommandListener()` method of the `Form` class to associate a listener with the form. Passing the `this` keyword to `setCommandListener()` specifies that the listener interface is implemented by the `MIDlet` class itself. The implementation of the one class in the interface, `commandAction()` is described below.

```

public void startApp()
{
    mEditBox = new TextField("Text Field Label:",null,MAXCHARS,0);
    mForm = new Form("MIDlet Developer Guide: Respond to information entered
in a text field.");
    mForm.append(mEditBox);

    mSubmitCmd = new Command("", "Submit", Command.OK, 0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);
}

```

- d. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```

public void startApp()
{
    mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
    mForm = new Form("MIDlet Developer Guide: Respond to information entered
in an edit box.");
    mForm.append(mEditBox);

    mSubmitCmd = new Command("", "Submit", Command.OK, 0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}

```

5. To retrieve the text entered in the text field, use the `getString()` method of the `TextField` instance stored in `mTextField`. Display it on the form by using the `append()` method of the `Form` class.

```
public void commandAction(Command c, Displayable disp)
{
    String strEnteredText = mTextField.getString();
    mForm.append("You entered " + strEnteredText);
}
```

Respond to option button selection

Option buttons are represented by an instance of the `ChoiceGroup` class with the `Choice.EXCLUSIVE` option specified. Selecting a particular radio button does not automatically result in an action. You have to associate and register a listener object to respond to a submitted selection.

You can create an `ItemStateListener` and register it to monitor any changes in the state of the items on a form. In the listener object, you could act upon the selection made by a user. This approach does not let the user change their initial selection. To provide the user with that option, create a separate submit command and a corresponding `CommandListener` instead of an `ItemStateListener`. Using this method, users can change their radio button selection until they are satisfied with their choice. Then they can use the submit command to provide their final choice. You respond to that final choice in the `commandAction()` method of the `CommandListener` interface.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`. Implement the `CommandListener` interface by including the `commandAction()` method.

```
public class RespondToOptionButtonSelection extends MIDlet implements
CommandListener
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }

    public void commandAction(Command c, Displayable disp)
    {
```

```

    }
}

```

3. Create private variables to store instances of the `Display`, `Form`, `ChoiceGroup`, and `Command` classes.

```

public class RespondToOptionButtonSelection extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;

    // MIDlet lifecycle method overrides omitted
}

```

4. In the overridden version of the `startApp()` method, create an instance of the `ChoiceGroup` object, populate it with option button choices and add it to a `Form` object.
 - a. Create an instance of the `ChoiceGroup` class to represent the set of option buttons. Specify `Choice.Exclusive` in the `ChoiceGroup` constructor to indicate that the group of choices are mutually exclusive and should, therefore, be presented as radio buttons. Use the `append()` method of the `ChoiceGroup` instance to create and add a label to each option button choice.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to radio button
selection",Choice.EXCLUSIVE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
}

```

- b. Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add to the form the set of radio buttons stored in the `mChoices` variable.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide:" +
        "Respond to radio button selection",Choice.EXCLUSIVE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Radio Button Selection");
    mForm.append(mChoices);
}

```

- c. Create a new instance of the `Command` class and store it in the private `mSubmitCmd` variable. Specify `Command.OK` as the second parameter in the constructor to indicate to the MIDP framework that the command should behave like an OK or submit button. Use the `addCommand()` method of the `Form` class to associate the command with `mForm`, the instance of the `Form` class that contains the checkboxes. Finally, use the `setCommandListener()` method of the `Form` class to associate a listener with the form. Passing the `this` keyword to `setCommandListener()` specifies that the listener interface is implemented by the `MIDlet` class itself. The implementation of the one class in the interface, `commandAction()` is described below.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Checkbox Selection");
    mForm.append(mChoices);

    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);
}
```

- d. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide:" +
        "Respond to radio button selection",Choice.EXCLUSIVE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Radio Button Selection");
    mForm.append(mChoices);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}
```

Respond to check box selection

To respond to check box selection on the BlackBerry device, you have to implement the `CommandListener` interface. You must also associate the listener with the form that contains the instance of the `ChoiceGroup` class that represents the check boxes. You make the association by using the `setCommandListener()` method of the `Form` object. Finally, you provide an implementation of the only method, `commandAction()`, in the `CommandListener` interface. In that implementation, you determine which check boxes the user selected and take appropriate action.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()` and `destroyApp()`. Implement the `CommandListener` interface by including the `commandAction()` method.

```
public class RespondToCheckBoxSelection extends MIDlet implements CommandListener
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }

    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

3. Create private variables to store instances of the `Display`, `Form`, `ChoiceGroup` and `Command` classes.

```
public class RespondToCheckBoxSelection extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;

    // MIDlet lifecycle method overrides omitted
}
```

4. In the overridden version of the `startApp()` method, create an instance of the `ChoiceGroup` object, populate it with check box choices and add it to a `Form` object.
- Create an instance of the `ChoiceGroup` class to represent the set of check boxes. Specify `Choice.MULTIPLE` in the `ChoiceGroup` constructor to indicate that the group of choices are mutually exclusive and should, therefore, be presented as check boxes. Use the `append()` method of the `ChoiceGroup` instance to create and add a label to each check box choice.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
}
```

- Create a new instance of the `Form` class. The constructor takes a string parameter to display as the title of the form. Use the `append()` method of the `Form` object to add to the form the set of check boxes stored in the `mChoices` variable.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Checkbox Selection");
    mForm.append(mChoices);
}
```

- Create a new instance of the `Command` class and store it in the private `mSubmitCmd` variable. Specify `Command.OK` as the second parameter in the constructor to indicate to the MIDP framework that the command should behave like an OK or submit button. Use the `addCommand()` method of the `Form` class to associate the command with `mForm`, the instance of the `Form` class that contains the checkboxes. Finally, use the `setCommandListener()` method of the `Form` class to associate a listener with the form. Passing the `this` keyword to `setCommandListener()` specifies that the listener interface is implemented by the `MIDlet` class itself. The implementation of the one class in the interface, `commandAction()` is described below.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
```

```

        mChoices.append("Third Choice", null);

        mForm = new Form("Respond to Checkbox Selection");
        mForm.append(mChoices);

        mSubmitCmd = new Command("Submit",Command.OK,0);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
    }

```

- d. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the form stored in the `mForm` variable.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Checkbox Selection");
    mForm.append(mChoices);

    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}

```

5. Create a boolean array, `arrSelected[]`, with the same number of elements as there are checkboxes. Call the `getSelectedFlags()` method of the `ChoiceGroup` object to populate the boolean array. After the method call, the array will contain element values of true in array positions that correspond to selected check boxes and element values of zero in array positions corresponding to unselected check boxes.

```

public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
        boolean arrSelected[] = new boolean[mChoices.size()];
        mChoices.getSelectedFlags(arrSelected);
    }
}

```

- To display a message that includes the names of each of the checkbox selections along with whether they were selected or not prior to submission, use a for loop to iterate through the array of boolean flags. Use the `getString()` method of the `ChoiceGroup` object to successively retrieve each check box label. Check the boolean array at the current index, `i`, to determine whether to display was selected or was not selected. Once the message string is constructed, display it on the form by using the `append()` method of the `Form` class.

```
public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
        boolean arrSelected[] = new boolean[mChoices.size()];
        mChoices.getSelectedFlags(arrSelected);
        for(int i = 0; i < mChoices.size(); i++)
        {
            String strMessage;
            strMessage = mChoices.getString(i);
            if(arrSelected[i])
            {
                strMessage += " was selected.";
            }
            else
            {
                strMessage += " was not selected.";
            }
            mForm.append(strMessage);
        }
    }
}
```

Respond to drop-down selection

- Import the following MIDlet libraries:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

- Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory MIDlet lifecycle methods: `startApp()`, `pauseApp()`, and `destroyApp()`.

```
public class RespondToDropDownSelection extends MIDlet implements CommandListener
{
    public void startApp()
    {
    }
}
```

```

public void pauseApp()
{
}

public void destroyApp(boolean flag)
{
}
}

```

3. Implement the `commandAction()` method of the `CommandListener` interface to respond to drop-down selection.

```

public class RespondToDropDownSelection extends MIDlet implements CommandListener
{
    // MIDlet lifecycle method overrides omitted

    public void commandAction(Command c, Displayable disp)
    {
    }
}

```

4. Create private variables to store instances of the `Display`, `Form`, `ChoiceGroup`, and `Command` classes.

```

public class RespondToDropDownSelection extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;

    // MIDlet lifecycle method overrides omitted
}

```

5. Create the drop-down.

- a. Use the `new` keyword to create an instance of the `ChoiceGroup` class. In the constructor, provide a title for the drop-down and specify `Choice.POPUP` to present the group of choices as a drop-down.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down Selection",Choice.POPUP);
}

```

- b. Use the `append()` method of the `ChoiceGroup` class to add items to the drop-down.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down Selection",Choice.POPUP);
}

```

```

    mChoices.append("First Choice");
    mChoices.append("Second Choice");
    mChoices.append("Third Choice");
}

```

6. Add the drop-down to a form.
 - a. Use the new keyword to create an instance of the Form class. In the constructor, provide a title for the form.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-
Down",Choice.POPUP);
    mChoices.append("First Choice");
    mChoices.append("Second Choice");
    mChoices.append("Third Choice");

    mForm = new Form("Respond to Drop-Down Selection");
}

```

- b. Use the append() method of the Form class to add the drop-down to the form.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);
}

```

7. Set up a submit command.
 - a. Use the new keyword to create an instance of the Command class. In the constructor, provide a caption for the command, the type of the command and its priority.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);
}

```

```

    mSubmitCmd = new Command("Submit",Command.OK,0);
}

```

- b. Use the `addCommand()` method of the `Form` class to associate the command with the form.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);

    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
}

```

- c. Pass the `this` keyword to the `setCommandListener()` method of the `Form` class to specify that the listener is implemented in the `RespondToDropDownSelection` class.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);

    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);
}

```

8. Display the form containing the drop-down.
- a. Use the static `getDisplay()` method of the `Display` class to retrieve an instance of the `Display` class that represents the current display.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
}

```

```

mChoices.append("Second Choice", null);
mChoices.append("Third Choice", null);

mForm = new Form("Respond to Drop-Down Selection");
mForm.append(mChoices);

mSubmitCmd = new Command("Submit",Command.OK,0);
mForm.addCommand(mSubmitCmd);
mForm.setCommandListener(this);

mDisplay = Display.getDisplay(this);
}

```

- b. Pass the `mForm` local variable to the `setCurrent()` method of the `Display` class to display the form on the screen.

```

public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);

    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);

    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}

```

9. Write code to respond to the selection of a drop-down list item.
- a. Implement the `commandAction()` method of the `CommandListener` interface.

```

public class RespondToDropDownSelection extends MIDlet implements
CommandListener
{
    // MIDlet lifecycle method overrides omitted

    public void commandAction(Command c, Displayable disp)
    {
    }
}

```

- b. Use an `if` statement to filter for the submit command.

```
public class RespondToDropDownSelection extends MIDlet implements
CommandListener
{
    // MIDlet lifecycle method overrides omitted

    public void commandAction(Command c, Displayable disp)
    {
        if(c == mSubmitCmd)
        {
        }
    }
}
```

- c. Use the `getSelectedIndex()` method to retrieve the index of the selected drop-down item.

```
public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
        int index = mChoices.getSelectedIndex();
    }
}
```

- d. Pass the index to the `getString()` method to retrieve the text associated with the selected drop-down item.

```
public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
        int index = mChoices.getSelectedIndex();
        String strChoiceText = mChoices.getString(index);
    }
}
```

- e. Use the `append()` method of the `Form` class to display a message on the screen indicating which drop-down item was selected.

```
public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
        int index = mChoices.getSelectedIndex();
        String strChoiceText = mChoices.getString(index);
        mForm.append("You selected " + strChoiceText);
    }
}
```

Respond to keyboard input

To respond to keyboard input, you must override the `keyPressed()` method of the `Canvas` class.

1. Import the two required MIDlet libraries.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Create the framework for the MIDlet by extending the `MIDlet` class and overriding the three mandatory methods: `startApp()`, `pauseApp()`, and `destroyApp()`.

```
public class RespondToKeyboard extends MIDlet
{
    public void startApp()
    {
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
}
```

3. Create private variables to store instances of the `Display`, `Form`, `ChoiceGroup` and `Command` classes.

```
public class RespondToKeyboard extends MIDlet
{
    private Display mDisplay;
    private KeyCanvas mKeyCanvas;

    // MIDlet lifecycle method overrides omitted
}
```

4. In the overridden version of the `startApp()` method, create an instance of your custom `KeyCanvas()` class. Use the static `getDisplay()` method of the `Display` class to retrieve a `Display` object that represents the current display. Store it in the `mDisplay` private variable. Use the `setCurrent()` method of the `Display` object to set the current displayable to the custom canvas.

```
public void startApp ()
{
    mKeyCanvas = new KeyCanvas();
}
```

```

    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mKeyCanvas);
}

```

5. Define an extension of the Canvas class. You must provide an implementation of the `paint()` method, but it does not have to do anything in this case. You override the `keyPressed()` method to respond to keyboard input. The sample uses `System.out.println()` to display a message in the output window indicating when a key with a positive key code is pressed.

```

class KeyCanvas extends Canvas
{
    public void paint(Graphics g)
    {
    }

    protected void keyPressed(int keyCode)
    {
        if (keyCode > 0)
        {
            System.out.println("You pressed " + ((char)keyCode));
        }
    }
};

```

Code sample: Respond to information entered in a text field

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class RespondToTextField extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private TextField mTextField;
    private Command mCommand;
    private static final int MAXCHARS = 100;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
        mForm = new Form("MIDlet Developer Guide: Respond to information entered in an
edit box.");
        mForm.append(mTextField);
        mCommand = new Command("", "Submit", Command.OK, 0);
        mForm.addCommand(mCommand);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }
}

```

```

}

public void pauseApp()
{
}

public void destroyApp(boolean flag)
{
}

public void commandAction(Command c, Displayable disp)
{
    String strEnteredText = mTextField.getString();
    mForm.append("You entered " + strEnteredText);
}
}

```

Code sample: Respond to radio button selection

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class RespondToRadioButtonSelection extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide:" +
            "Respond to radio button selection",Choice.EXCLUSIVE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);

        mSubmitCmd = new Command("Submit",Command.OK,0);

        mForm = new Form("Respond to Radio Button Selection");
        mForm.append(mChoices);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }

    public void destroyApp(boolean flag)
    {

```

```

}

public void pauseApp()
{
}

public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
        int index = mChoices.getSelectedIndex();
        String strChoiceText = mChoices.getString(index);
        mForm.append("You selected " + strChoiceText);
    }
}
}

```

Code sample: Respond to check box selection

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class RespondToCheckBoxSelection extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);

        mSubmitCmd = new Command("Submit",Command.OK,0);

        mForm = new Form("Respond to Checkbox Selection");
        mForm.append(mChoices);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }

    public void destroyApp(boolean flag)
    {

```

```

}

public void pauseApp()
{
}

public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
        boolean arrSelected[] = new boolean[mChoices.size()];
        mChoices.getSelectedFlags(arrSelected);
        for(int i = 0; i < mChoices.size(); i++)
        {
            String strMessage;
            strMessage = mChoices.getString(i);
            if(arrSelected[i])
            {
                strMessage += " was selected.";
            }
            else
            {
                strMessage += " was not selected.";
            }
            mForm.append(strMessage);
        }
    }
}
}
}

```

Code sample: Respond to drop-down selection

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class RespondToDropDown extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;

    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to drop-
down",Choice.POPUP);
        mChoices.append("First Choice", null);
    }
}

```

```

        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);

        mForm = new Form("Display a Drop-down");
        mForm.append(mChoices);
        mSubmitCmd = new Command("", "Submit", Command.OK, 0);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);

        mDisplay.setCurrent(mForm);
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }

    public void commandAction(Command c, Displayable disp)
    {
        if(c == mSubmitCmd)
        {
            boolean arrSelected[] = new boolean[mChoices.size()];
            mChoices.getSelectedFlags(arrSelected);
            for(int i = 0; i < mChoices.size(); i++)
            {
                String strMessage;
                strMessage = mChoices.getString(i);
                if(arrSelected[i])
                {
                    strMessage += " was selected.";
                }
                else
                {
                    strMessage += " was not selected.";
                }
                mForm.append(strMessage);
            }
        }
    }
}

```

Code sample: Respond to keyboard input

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

```

```
public class RespondToKeyboard extends MIDlet
{
    private Display mDisplay;

    public void startApp ()
    {
        mDisplay = Display.getDisplay(this);
        KeyCanvas mKeyCanvas = new KeyCanvas();
        mDisplay.setCurrent(mKeyCanvas);
    }

    public void pauseApp()
    {
    }

    public void destroyApp(boolean flag)
    {
    }
};

class KeyCanvas extends Canvas
{
    public void paint(Graphics g)
    {
    }

    protected void keyPressed(int keyCode)
    {
        if (keyCode > 0)
        {
            System.out.println("You pressed " + ((char)keyCode));
        }
    }
};
```

Glossary

6

API

application programming interface

JVM

Java® Virtual Machine

Java ME

Java® Platform, Micro Edition

JSR

Java® Specification Request

MIDP

Mobile Information Device Profile

Provide feedback

7

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

Document revision history

8

Date	Description
8 February 2010	<p>Updated the following topic to mention that devices that run BlackBerry® Device Software version 5.0 or later support MIDP 2.1.</p> <ul style="list-style-type: none"><li data-bbox="429 453 846 489">• Java ME and Java APIs for BlackBerry

Legal notice

9

©2010 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.

Java and Java ME are trademarks of Sun Microsystems, Inc. All other trademarks are the property of their respective owners.

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