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 those provided or made available by hyperlink 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 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 AFFILIATE 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 may not offer Internet browsing functionality with a subscription to 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, BlackBerry® Device Software and/or BlackBerry Handheld Software and may require additional development or Third Party Products and Services for access to corporate applications.

Certain products mentioned in this documentation include software developed by the Apache Software Foundation (http://www.apache.org/) and/or licensed pursuant to Apache License, Version 2.0 (http://www.apache.org/licenses/). For more information, see the NOTICE.txt file included with the software. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

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.
Chapter 3

Developing a user interface for mobile devices

Objectives

- Describe MIDP
- Describe the BlackBerry® CLDC application
- Explain the best practices for successful mobile device UIs
- List and explain the components of the BlackBerry smartphone UI
- Explain how to create a basic UI for a mobile application for a mobile device such as a BlackBerry smartphone
- Describe how to evaluate a basic BlackBerry smartphone UI.

This chapter outlines how to develop a basic UI. This chapter describes the MIDP, and the BlackBerry CLDC application. This chapter outlines best practices for successful mobile device UIs. This chapter lists and explains components of the BlackBerry smartphone UI. This chapter describes how to create and evaluate a basic BlackBerry smartphone UI.
Introducing the BlackBerry smartphone UI

The main structure for a BlackBerry smartphone UI is the `Screen` object. A BlackBerry Java® Application can display more than one screen at a time, but only one screen in a BlackBerry Java application is active at one time.

A graphical UI manages the relationship between the application and its user, while providing as much information as possible in an easy to read manner using available screen space. An application that is easy to use lets a user perform tasks in an efficient manner, whereas a poor design can easily frustrate that person and prevent them from using the application. A graphical UI must feel intuitive and familiar to a user.

You must select your UI APIs according to the type of application you want to create. If you design a UI for a BlackBerry Java® Application, you use the BlackBerry UI APIs. If you design MIDlets, use the MIDP UI APIs.

A BlackBerry® Application is an application that uses the BlackBerry APIs, as well as the CLDC APIs and possibly the MIDP APIs. A BlackBerry API application extends the `UiApplication` class and starts with a standard `main()` method.

Most of the BlackBerry® Java® Development Environment sample applications are BlackBerry API applications. All the BlackBerry Device Software applications (including the message list, contact list, calendar, and BlackBerry® Browser) are BlackBerry API applications.

Advantages of using the BlackBerry APIs include the following:

- BlackBerry UI APIs provide more functionality and flexibility than the standard `javax.microedition.lcdui` APIs that you use to create MIDlets.
- Applications can run background threads after the application closes.
- Applications can start automatically in the background when the device turns on.
- Applications can use IPC APIs to exchange information with other applications.
- Developers can create library modules that BlackBerry API applications can import.
- Applications can access trackball and trackwheel events.
- Applications can access touch screen events on a BlackBerry device with a touch screen.
- Applications can use data from the accelerometer on a BlackBerry device.
- Applications can integrate with BlackBerry Browser, BlackBerry® Maps, and other applications.
- Applications can change the icon that displays on the Home screen on a BlackBerry device.

The only disadvantage of using BlackBerry APIs is that a BlackBerry API application can run only on a BlackBerry device.
Explaining best practices for successful mobile device UIs

Applications designed for BlackBerry smartphones can provide a balance between the best possible user experience and a long battery life.

When you design your BlackBerry smartphone application, consider the following limitations of mobile devices:

- Mobile devices have a small screen size that can display a limited number of characters.
- Mobile devices have slow processor speeds.
- Mobile devices use wireless networks that have a longer latency period than standard LANs.
- Mobile devices have a small amount of available memory.
- Mobile devices have a short battery life.
- Mobile devices display one screen at a time.

Mobile device users use applications on mobile devices differently than on computers. On mobile devices, users expect to find information quickly. For example, a customer relationship management system can provide a massive amount of information, but users only require a small amount of that information at one time. The BlackBerry smartphone UI is designed so that users can perform tasks easily and access information quickly.

Before you design your application, consider using the core applications on the BlackBerry smartphone or the BlackBerry Smartphone Simulator to learn more about the navigation model and best practices for designing the UI for your application.

Best practices for designing applications for BlackBerry smartphones

When you design applications for BlackBerry smartphones, try to be as consistent as possible with other BlackBerry smartphone applications.

Consider the following guidelines:

- Use or extend existing UI components where possible so that your application can inherit the default behavior of the component.
- Follow the standard navigation model as closely as possible so that a particular user action produces a consistent result across applications.
- Support and extend user tasks in useful ways. For example, after users download an application, the application can open automatically. The application can be saved in the Applications folder.

When you design your application, also consider the following guidelines:

- Stay focused on users’ immediate task. Simplify data selection and presentation by displaying only the information that users need at any one moment.
- Verify that the actions available in the menu are relevant to users’ current context.
• Minimize the number of times that users need to click the trackwheel, trackball, trackpad, or touch screen to complete a task.
• Design your UI to allow users to undo commands. For example, always display a dialog box if users click a destructive menu item or button. Users sometimes click the wrong menu item or button accidentally.
• Display information in a way that makes effective use of the small screen.
Components of the BlackBerry smartphone UI

The BlackBerry UI APIs are a library of UI components that are designed to provide default layouts and behaviors that are consistent with the core BlackBerry smartphone applications. The BlackBerry graphical user interface (GUI) in its most basic form is a three-level hierarchical structure, in which Screen classes contain Manager classes, and Manager classes contain Field classes.

Screen components provide a standard screen layout, a default menu, and standard behavior when the BlackBerry smartphone user presses the Escape key or clicks the trackwheel or trackball. The Screen class (net.rim.device.api.ui.Screen) is the starting point for the BlackBerry GUI. The BlackBerry smartphone displays only one screen at a time. The BlackBerry smartphone displays screens by pushing and popping them off of the display stack. The BlackBerry smartphone shows the user the screen on the top of the display stack. A screen can only exist once in the display stack, but the BlackBerry smartphone can push or pop it from the display stack at any time. The net.rim.device.api.ui.container package contains common sub classes of the Screen class.

Manager components provide an application with the ability to arrange components on a BlackBerry smartphone screen in standard ways, such as horizontally, vertically, or in a left-to-right flow. The Manager class (net.rim.device.api.ui.Manager) manages the layout and interaction between field objects you place in it. With a manager you can specify the location and layout of fields it contains, scrolling, and focus change between fields. Every screen must contain at least one manager. The net.rim.device.api.ui.container package contains common sub classes of the Manager class.

Field components provide standard UI elements for date selection, options, check boxes, lists, text fields and labels, and progress bar controls. The Field class provides the fundamental functionality for all field components. A field is a rectangular region that displays output to a user. A field can also handle input, and you must choose if you want to show the input to a user. The net.rim.device.api.ui.component package contains a library of prebuilt GUI components and controls that you can use to construct BlackBerry applications.

You can use the BlackBerry UI APIs to create UIs that include tables, grids, graphics, and other specialized features. The BlackBerry® Java® Development Environment uses a standard Java event model to receive and respond to specific types of events. Applications can receive and respond to BlackBerry smartphone user events, such as when the BlackBerry smartphone user clicks the trackwheel, clicks the trackball, or types on the keyboard, and to system events, such as global alerts, real-time clock changes, and USB port connections.

Note:
You can always find more information about the APIs we are using in the BlackBerry API reference document that is the part of the JDE Component Pack. You can find it on your computer under Start > Programs > Research in Motion > BlackBerry JDE 4.x.x.
1. Which of the following UI elements controls the way elements are arranged on a BlackBerry smartphone?
   
   A. Screen component
   B. Field component
   C. Layout manager

2. Which of the following APIS can you use in a BlackBerry API application?
   
   A. net.rim.device.api.ui.Screen
   B. net.rim.device.api.ui.Manager
   C. net.rim.device.api.ui.component.ButtonField
   D. All of the above
Answers

1. C
2. D
Chapter 3

Create a basic UI for a mobile application

You can use the hierarchy of BlackBerry GUI classes in which Screen classes contain Manager classes, and Manager classes contain Field classes to create a basic UI for a mobile application.

Screens

The main structure for a BlackBerry smartphone UI is the Screen object. A BlackBerry Java application can display more than one screen at a time, but only one screen in a BlackBerry Java application is active at one time.

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

The Screen class does not implement disambiguation, which is required for complex input methods, such as international keyboards and the BlackBerry 7100 Series. For seamless integration of the different input methods, extend Field or one of its subclasses. Do not use Screen objects for typing text.

To create a screen, do the following:

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.

<table>
<thead>
<tr>
<th>Screen type</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Screen</td>
<td>Use the Screen 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 Field superclass define.</td>
</tr>
</tbody>
</table>
When designing the UI for your applications, you must consider both the screen size and screen display methods of BlackBerry smartphones.

### Screen size

BlackBerry smartphones have a small screen size and can display a limited number of characters. BlackBerry smartphone screens also have little space for icons and other screen components.

You must test your application UI to ensure that it displays properly on different BlackBerry smartphone models. Also, on BlackBerry smartphones with a touch screen, users can switch between portrait view and landscape view by turning the device sideways.

### Screen display

BlackBerry smartphones display one screen at a time because they have a small screen size.
Common UI elements that you can display on the BlackBerry smartphone include the following:

- application screens
- dialog boxes
- progress indicators

### Application screens

A BlackBerry smartphone can have multiple screens open at the same time, but users can only interact with one screen in a BlackBerry Java application at one time. If users open multiple screens, the BlackBerry smartphone organizes the screens in a stack. The screen at the top of the stack is the active screen. If an application displays a screen, the BlackBerry® Java® Virtual Machine pushes it to the top of the stack. When an application closes a screen, the BlackBerry Java Virtual Machine deletes the screen from the top of the stack and displays the next screen on the stack, redrawing it as necessary.

An application screen can include the following elements: nonscrolling title bar, scroll arrows or bars, context menu, full menu, and shortcut bar.

<table>
<thead>
<tr>
<th>Screen element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonscrolling title bar</td>
<td>This bar appears at the top of the screen and can include a descriptive title of the screen.</td>
</tr>
<tr>
<td>scroll arrows or bars</td>
<td>If content extends beyond the viewing area, scroll arrows or scroll bars appear. Vertical scroll arrows or scroll bars appear on the right side of the screen, and horizontal scroll bars appear at the bottom of the screen. Scroll bars do not interfere with text.</td>
</tr>
<tr>
<td>context menu</td>
<td>A context menu contains a list of the most common available actions that users can perform within the current context.</td>
</tr>
<tr>
<td></td>
<td>On BlackBerry smartphones with a trackball or trackpad, a context menu appears if users click the trackball or trackpad and there is no default action. The context menu appears at the bottom of a screen, centered horizontally.</td>
</tr>
<tr>
<td></td>
<td>On BlackBerry smartphones with a touch screen, a context menu appears if users click an item that has a number of common actions associated with it (for example, a date separator in a message list or a contact name in a message). The context menu appears to the right of the highlighted item. If there is not enough space to the right, the context menu appears to the left of the highlighted item.</td>
</tr>
<tr>
<td>full menu</td>
<td>A full menu contains all the available actions for the application. To open a full menu, users press the Menu key. Full menus appear in the lower-left corner of the screen.</td>
</tr>
</tbody>
</table>
Dialog boxes

Use dialog boxes to perform the following actions:

- Prompt users for information that the application requires to complete a user-initiated task.
- Inform users of urgent information or the status of important actions.
- Warn users of unexpected or potentially destructive conditions or situations.

A dialog box includes an indicator that indicates the type of dialog box, a message, and buttons that allow users to perform an action. The size of the dialog box depends on the size of the BlackBerry smartphone screen. Scroll arrows appear if necessary. The theme that users select on their BlackBerry smartphone determines the visual style of the dialog box.

Different types of dialog boxes include OK, Delete, Save, Cancel, Yes/No.

To create a dialog box, do the following:

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.

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

Progress indicators

Use progress indicators to indicate the status of an operation. The two types of progress indicators include definite and indefinite progress indicators.

- Use definite progress indicators if you can determine the duration of an operation. Definite progress indicators include a label to indicate what the operation is and a horizontal bar that fills from left to right as an operation progresses. A percentage appears in the bar to indicate how much of the operation is complete. In the browser, progress indicators also indicate the number of kilobytes out of a total number of kilobytes that a BlackBerry smartphone downloads. To hide a definite progress indicator, users press the End key. If users press the End key, the operation continues, but users can perform other tasks at the same time.

- Use an indefinite progress indicator if you cannot determine the duration of an operation. You can indicate progress using a status dialog box. Users cannot perform other actions within the application while an application displays an indefinite progress indicator.

To create a progress indicator, do the following:

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

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

**Menus**

The Application Menu Item API, in the `net.rim.blackberry.api.menuitem` package, allows you to add menu items to BlackBerry Java applications. The `ApplicationMenuItemRepository` class allows you to add or remove menu items from BlackBerry Java applications.

With this code, you can create menus that users can use to perform many of the actions in an application. The full menu includes all the actions that users can perform in the application. After users press the Menu key, the full menu appears in the lower-left corner of the screen. Scroll arrows appear if the application offers more menu items that the user cannot see on the screen.

To create a menu, do the following:

1. Import the required classes and interfaces.
   ```java
   import net.rim.device.api.ui.*;
   import net.rim.device.api.ui.component.*;
   import net.rim.device.api.ui.container.*;
   ```

2. Create the application framework by extending the `UiApplication` class. In `main()`, create an instance of the new class and invoke `enterEventDispatcher()`, to enable the application to receive events. In the constructor, invoke `pushScreen`, to display the custom screen for the application. The `CreateMenuScreen` class represents the custom screen.

   ```java
   public class CreateMenu extends UiApplication {
   public static void main(String[] args) {
   CreateMenu theApp = new CreateMenu();
   theApp.enterEventDispatcher();
   }
   public CreateMenu() {
   pushScreen(new CreateMenuScreen());
   ```
3. Create the custom screen for the application by extending the `MainScreen` class. In the constructor, invoke `setTitle()`, to display the title for the screen. Invoke `add()`, to display a text field on the screen. Invoke `addMenuItem()`, to add a menu item to the default menu that is included with `MainScreen`.

```java
class CreateMenuScreen extends MainScreen {
    public CreateMenuScreen() {
        setTitle("Create Menu Sample");
        add(new RichTextField("Create a menu"));
        addMenuItem(_viewItem);
    }
}
```

4. Create the menu item by using the `MenuItem` class. Override `run()`, to specify the action that occurs when the user clicks the menu item. When the user clicks the menu item, the application invokes `Menu.run()`.

```java
private MenuItem _viewItem = new MenuItem("More Info", 110, 10) {
    public void run() {
        Dialog.inform("Display more information");
    }
};
```

5. Override `close()`, to display a dialog box to the user when the user clicks the Close menu item. The Close menu item is included by default. Invoke `super.close()`, to close the application. When the user closes the dialog box, the application invokes `MainScreen.close()`, to close the application.

```java
public void close() {
}
```
{  
    Dialog.alert("Goodbye!");  
    super.close();  
}

**Context menus**

You can create a context menu, or short menu, which contains a list of the most common available actions that users can perform within the current context.

BlackBerry smartphones that have a trackball and menu key use two menu categories: primary action menus and full menus. Users can click a trackball to perform one of two actions: execute the default action (such as opening an email message), or display the primary actions menu (short menu). Users can press the menu key to display the full menu (default menu), which contains all menu items for the current screen. To build a primary actions menu, you must place a check in the `makeMenu()` method so the application can determine which menu to invoke.
1. Which of the following objects is the main structure for a BlackBerry smartphone UI?
   A. Manager
   B. Screen
   C. Menu

2. Which of the following classes provides features that are common to standard BlackBerry Java applications?
   A. Screen
   B. FullScreen
   C. MainScreen

3. Which of the following statements about BlackBerry smartphone with a touch screen is false?
   A. Has limited space for text, icons, and other screen components
   B. Can display multiple screens at the same time
   C. Can display in both portrait or landscape mode

4. Which of the following items appears only on BlackBerry smartphones with a touch screen?
   A. Context menu
   B. Full menu
   C. Shortcut bar
   D. Scroll arrows

5. Which actions does a context menu display in an application?
   A. All the actions
   B. The primary actions
   C. The most commonly used actions
   D. The most recently used actions
Answers

1. B
2. C
3. B
4. C
5. C
Managers

The Manager class is the next level in the BlackBerry GUI hierarchy. Managers build on top of screens. You use managers for scrolling (vertical and/or horizontal) and to position and lay out fields. You add fields to managers to place them in an appropriate area on the screen. The BlackBerry API set contains a set of managers that extend from the Manager class and provide a layout mechanism for common screen designs. Managers include the following:

- **HorizontalFieldManager**: A manager that lays out fields from left to right in a single row. This manager can provide horizontal scrolling for fields that do not fit on the screen as well as vertical scrolling for fields that are taller than the screen.
- **VerticalFieldManager**: A manager that lays out fields in a single vertical row. This manager can provide vertical scrolling for fields that do not fit on the screen as well as horizontal scrolling for fields that are wider than screen.
- **FlowFieldManager**: A manager that lays out fields in a horizontal then vertical flow. Fields are positioned from left to right. This manager places any fields that do not fit in the allotted horizontal space on the next line below, beginning from the left. The FlowFieldManager also supports both horizontal and vertical scrolling.
- **DialogFieldManager**: This manager handles an icon, a message, and a special area, which can hold a list of user-specified custom fields. Use a VerticalFieldManager to lay out the fields in the user area. A VerticalFieldManager lays out its icon in the top left corner, and its message label in the top left corner.

To create a layout manager, do the following:

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

3. Invoke the add() method to add the **HorizontalFieldManager** to a screen.
   
   ```java
   myScreen.add(_fieldManagerBottom);
   ```

4. Create an instance of a **ButtonField**.
   
   ```java
   ButtonField mySubmitButton = new ButtonField("Submit");
   ```

5. Add the **ButtonField** to the **HorizontalFieldManager**.
   
   ```java
   _fieldManagerBottom.add(mySubmitButton);
   ```
Nesting managers

Managers are an extension of the Field class. This means managers can also contain other managers. This is known as nesting. You can use nesting to create various enhanced layout styles on the BlackBerry smartphone; such as a column or table layout.

If you nest one manager into another manager, you can create various field layouts on the screen. If you select managers that enable scrolling, you can also control how a user interacts with the fields on the screen.

Fields

Field components provide standard UI elements for controls such as date selection, options, check boxes, lists, text fields and labels, and progress bars.

Fields represent all UI components, which are rectangular regions that a Manager contains. A field’s layout requirements determine the size of the field. Managers provide scrolling for the fields that they contain.

To create a specialized field component (for example, a text field that contains multiple elements), create your own custom types by extending the Field class or one of its subclasses.

<table>
<thead>
<tr>
<th>Traditional field</th>
<th>BlackBerry Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>ButtonField</td>
</tr>
<tr>
<td>Check box</td>
<td>CheckboxField</td>
</tr>
<tr>
<td>Date</td>
<td>DateField</td>
</tr>
<tr>
<td>Dialog box</td>
<td>PopupScreen</td>
</tr>
<tr>
<td>Drop-down list</td>
<td>NumericChoiceField or ObjectChoiceField</td>
</tr>
<tr>
<td>Radio button</td>
<td>RadioButtonField</td>
</tr>
<tr>
<td>Text</td>
<td>RichTextField, BasicEditField, EditField, PasswordEditField, or AutoTextBoxEditField</td>
</tr>
<tr>
<td>Text label</td>
<td>LabelField</td>
</tr>
<tr>
<td>List</td>
<td>ListField</td>
</tr>
</tbody>
</table>

When designing controls for the BlackBerry smartphone UI, you must select an appropriate control and ensure that the control works with different user interaction methods.
Developing a user interface for mobile devices

Buttons

When you add this code, an action occurs after users click a button.

To create a button, do the following:

1. **Import the** `net.rim.device.api.ui.component.ButtonField` **class.**
2. **Create an instance of a** `ButtonField using a style parameter.**
   
   ```java
   ButtonField mySubmitButton = new ButtonField("Submit");
   ```

Check boxes

With this code, you can create check boxes for options that users can turn on or turn off.

To create check boxes, do the following:

1. **Import the** `net.rim.device.api.ui.component.CheckboxField` **class.**
2. **Create an instance of a** `CheckboxField.**
   
   ```java
   CheckboxField myCheckbox = new CheckboxField();
   ```

Drop-down lists

With this code, you can create drop-down lists to provide a set of mutually exclusive values.

To create drop-down lists, do the following:

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

   ```java
   String choiceItems[] = {"Option one","Option two","Option three"};
   ObjectChoiceField myObjectChoiceField = new ObjectChoiceField("Pick One: ", choiceItems);
   ```

List boxes

With this code, you can create a list box to display a list from which users can select one or more values.
To create list boxes, do the following:

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.
   
   ```java
   private class ListCallback implements ListFieldCallback {
   
   4. Create the items that you want to display in a ListField.
      
      String fieldOne = new String("Marco Cacciacarro");
      String fieldTwo = new String("Meredith Wagler");

   5. Create an instance of a ListField.
      
      ListField myList = new ListField(0, ListField.MULTI_SELECT);

      
      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);
   
   Option buttons

   With this code, you can create option buttons to indicate a set of mutually exclusive but related choices.

   To create option buttons, do the following:

   1. Import the following classes:
Developing a user interface for mobile devices

- net.rim.device.api.ui.component.RadioButtonGroup
- net.rim.device.api.ui.component.RadioButtonField


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

3. Create an instance of a RadioButtonField for each option you want to make available to the BlackBerry® device user.

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


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

Text fields

With this code, you can create a text field in which users can type text.

Create a read-only text field that allows formatting

To create a read-only text field that allows formatting, do the following:

1. Import the `net.rim.device.api.ui.component.RichTextField` class.

2. Create an instance of a RichTextField.

```
RichTextField rich = new RichTextField("The text I want to show.");
```

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

To create an editable text field that has no formatting and accepts filters, do the following:

1. Import the following class:
   - net.rim.device.api.ui.component.BasicEditField

2. Create an instance of a BasicEditField.
BasicEditField bf = new BasicEditField("BasicEditField: ", "", 10, EditField.FILTER_UPPERCASE);

Create an editable text field that allows special characters

To create an editable text field that allows special characters, do the following:

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 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.

To create a text field for AutoText, do the following:

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: ", "");

Search fields

With this code, you can create search fields to allow users to search for items in an application. Several BlackBerry smartphone applications include search fields. For example, in the task list, users can use a search field to search for specific tasks.

As users type text in a search field, the application searches for and displays items that begin with the search text. If users type multiple words in the search field, the application displays results that match both words. For example, if users type “ca ba”, the application returns all items with a word that begins with “ca” and a word that begins with “ba” (such as “call back”).

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.
To create search fields, do the following:

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.**

   ```java
   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.**

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

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

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

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

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

7. **Invoke** KeywordFilterField.setList().

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

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

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

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

   ```java
   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

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

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

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

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

    _keywordField.updateList();

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

    String userTypedWord = _keywordField.getKeyword

Tree views

With this code, you can create a tree view to display objects, such as a folder structure, in a hierarchical manner.

Objects in the tree view are 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.

To create a field to display a tree view, do the following:

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.

```java
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);
```

5. 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.

```java
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);
    }
}
Developing a user interface for mobile devices
1. Which of the following classes provides an application with the ability to arrange components on a BlackBerry device screen?
   A. Screen
   B. Manager
   C. Field

2. Which control can you use to initiate an action?
   A. Check box
   B. Button
   C. Drop-down list

3. Which type of content do list boxes display?
   A. A set of mutually exclusive but related values
   B. A set of mutually exclusive values
   C. A set of values

4. Which type of control can you use to display objects in a hierarchical structure?
   A. Tree view
   B. List box
   C. Search field
   D. Any of the above
Answers

1. B
2. B
3. C
4. A
Drawing directly on the screen

There are two main areas where you can typically use the Graphics class: at the field level and the screen level. You can draw at the field level to create a unique field such as a custom looking button or a graph. You can draw at the screen level to have full control of the screen and what is presented to the BlackBerry smartphone user. Typically you draw at the screen level to make use of animation or multimedia content in an application, such as games and graphic demos.

Graphics class

Whether you create a custom screen or a custom field, you perform the bulk of the work of the Graphics class in the paint method of the Field class (remember that the Screen class extends the Field class). A BlackBerry smartphone invokes the paint method when a field, or part of the field, must be redrawn.

You can override the paint method to create a custom field, or to modify an existing field. One of the simplest things you can do by overriding the paint method is to change the color used. For example you can change the text color of a RichTextField to green by doing the following:

```java
//Green - The format for color is 0x00RRGGBB.
Long myColor = 0x00008800;

RichTextField colorChange =
    new RichTextField("The quick brown fox jumps over the lazy dog.");
{
    public void paint(Graphics graphics)
    {
        //Change the color of the text in
        //the RichTextField to green.
        graphics.setColor(myColor);
    }
}
```

If your application does not contain any fields, you can override the `javax.microedition.lcdui.Canvas.paint(Graphics)` method to perform the painting using the specified Graphics object.
//call super.paint() to carry out the
//default painting of the field
super.paint(graphics);
}

You can also change values used within the paint method between calls to paint to create animation. Continuing from the preceding code sample, you can do the following:

//Change the color to red.
myColor = 0x00FF0000;
colorChange.invalidate();

You can force a field, or part of a field, to be redrawn by invoking one of its invalidate methods. If you call the invalidate() method, you mark the entire field as invalid by instructing the entire field to be redrawn. If you call invalidate(int x, int y, int width, int height), you mark a certain region of a field as invalid. By marking a certain region of a field as invalid, you can increase the speed of the painting because only a certain area of a field has changed. Only the region specified is redrawn, which can provide a substantial performance increase for large fields, or fields that are redrawn in rapid succession such as when performing animation.

To draw in a specific bitmap, a BlackBerry smartphone application obtains a graphics context by passing the bitmap object into the Graphics constructor. To draw over a bitmap without changing the content of the bitmap, create a BitmapField and pass the Bitmap object into the BitmapField constructor. When drawing in a field, the field manager passes a graphics context to the fields when the fields repaint. To perform custom drawing in a field, override the paint(Graphics g) method when you extend the Field class. Use the Graphics class to draw shapes, such as arcs, lines, rectangles, and circles.

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.

The following code sample illustrates how you can override the paint() method to draw a colored circle on the screen. In the following code sample, the getWidth() and getHeight() methods are used to retrieve the dimensions of the screen on the BlackBerry smartphone 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);
    }

}

The following sections detail how to draw simple shapes to the screen of a BlackBerry smartphone.
Drawing a circle

To draw a circle on the BlackBerry smartphone 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()`.

```java
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.

```java
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.

```java
public void startApp()
{
    Displayable disp = new CircleCanvas();
    Display.getDisplay(this).setCurrent(disp);
}
```
Drawing a rectangle

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

Import the two required MIDlet libraries. `import javax.microedition.midlet.*;` and `import javax.microedition.lcdui.*;`.

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

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

    public void pauseApp()
    {
    }

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

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

```java
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);

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);
}

Drawing graphics within graphics

You can also create graphics within graphics. Vary the approach you take to create your graphics based on their usage pattern. You can successfully override the paint method for a field or screen that changes often, such as during an animation, but this approach is less desirable when you are creating an image that rarely changes but is reused throughout an application. The following code sample is an example of how you can create a dynamic bitmap you can use to populate a BitmapField. The following code sample shows a bitmap that contains a large purple square and a green circle:

    //Instantiate a bitmap 100x100 pixels in size.
    Bitmap myBitmap = new Bitmap(100, 100);

    Graphics myGraphics = new Graphics(myBitmap);

    //Change the colour to purple.
myGraphics.setColor(0x00550055);

//Draw a filled rectangle starting at 0,0
//that is 100 pixels square.
myGgraphics.fillRect(0, 0, 100, 100);

//Change the colour to green.
myGraphics.setColor(0x00004400);

//Draw a filled circle that has a radius of 25.
myGraphics.fillArc(50, 50, 25, 25, 0, 360);

//Instantiate a BitmapField with the
//created bitmap.
BitmapField myBitmapField = new BitmapField(myBitmap);

You can use this technique to store the bitmap and display it again in other areas at a later time. If the image is fairly detailed and requires multiple calls to several of the graphics methods, you can save some processing time by creating the image once and reusing it. It is faster to reuse and image than to recreate an image every time because you use graphics methods from within the `paint` method and reuse the existing bitmap image.

Although ideal for images that remain fairly constant, this approach is not optimal for highly dynamic content.
1. Which method is most commonly used to create graphics in the Graphics class?

2. Which method do you use when something must be redrawn?

3. Which class do you extend and which method do you override, typically, to customize graphics?
Answers

1. Paint method

2. Invalidate method

3. Extend the Canvas class and override the paint method.
Create custom buttons

Example: CustomButtonField.java

/**
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.custombuttons;
import net.rim.device.api.ui.*;
import net.rim.device.api.system.*;

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

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

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

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

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

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

/* Method that draws the focus indicator for this button and inverts the inside region of the shape. */
protected void drawFocus(Graphics graphics, boolean on) {
    switch(_shape) {

case TRIANGLE:
    int w = getWidth();
    int h = w >> 1;
    for (int i=h-1; i>=2; --i) {
        graphics.invert(i, h - i, w - (i << 1), 1);
    }
    break;

case RECTANGLE:
    graphics.invert(1, 1, getWidth() - 2, getHeight() - 2);
    break;

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

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

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

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

/* Sets the shape. */
public void setShape(int shape) {
    _shape = shape;
    updateLayout();
}

/* Retrieves the preferred width of the button. */
public int getPreferredWidth() {
    switch(_shape) {
    case TRIANGLE:
        if (_labelWidth < _labelHeight) {
            return _labelHeight << 2;
        } else {
            return _labelWidth << 1;
        }
    case SQUARE:
        return _labelWidth;
    case CIRCLE:
        return _labelHeight;
    case DIAMOND:
        return Math.max(_labelWidth, _labelHeight);
    default:
        return 0;
    }
}
case OCTAGON:
    if (_labelWidth < _labelHeight) {
        return _labelHeight + 4;
    } else {
        return _labelWidth + 8;
    }

case RECTANGLE: default:
    return _labelWidth + 8;
}

/* Retrieves the preferred height of the button. */
public int getPreferredSize() {
    switch(_shape) {
    case TRIANGLE:
        if (_labelWidth < _labelHeight) {
            return _labelHeight << 1;
        } else {
            return _labelWidth;
        }
    case RECTANGLE:
        return _labelHeight + 4;
    case OCTAGON:
        return getPreferredSize();
    }
    return 0;
}
Developing a user interface for mobile devices

/* Lays out this button’s contents.
* This field’s manager invokes this method during the layout
* process to instruct this field to arrange its contents, given an
* amount of available space.
**/

protected void layout(int width, int height) {
    // Update the cached font in case it has been changed.
    _font = getFont();
    _labelHeight = _font.getHeight();
    _labelWidth = _font.getAdvance(_label);

    // Calculate width.
    width = Math.min( width, getPreferredWidth() );

    // Calculate height.
    height = Math.min( height, getPreferredHeight() );

    // Set dimensions.
    setExtent( width, height );
}

/*
* Redraws this button. The field’s manager invokes this method during
* the repainting process to instruct this field to repaint itself.
*/

protected void paint(Graphics graphics) {
    int textX, textY, textWidth;
    int w = getWidth();
    switch(_shape) {
        case TRIANGLE:
int h = (w>>1);
int m = (w>>1)-1;
graphics.drawLine(0, h-1, m, 0);
graphics.drawLine(m, 0, w-1, h-1);
graphics.drawLine(0, h-1, w-1, h-1);
textWidth = Math.min(_labelWidth,h);
textX = (w - textWidth) >> 1;
textY = h >> 1;
break;

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

case RECTANGLE: default:
    graphics.drawRect(0, 0, w, getHeight());
Developing a user interface for mobile devices

textX = 4;
textY = 2;
textWidth = w - 6;
break;
}
graphics.drawText(_label, textX, textY, (int)( getStyle() &
DrawStyle.ELLIPSIS | DrawStyle.HALIGN_MASK ),
textWidth );
}

Create a custom context menu

Example: ContextMenuSample.java

/**
 * ContextMenuSample.java
 *
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights
 * reserved.
 */

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

public class ContextMenuSample extends UiApplication implements ContextMenuSampleResource
private MyContextField myContextField;

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

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

// Inner class to define a new field.

private static class MyContextField extends RichTextField {
    private MenuItem myContextMenuItemA = new MenuItem(
        _resources, MENUITEM_ONE, 200000, 10) {
        public void run() {
            onMyMenuItemA();
        }
    };

    private MenuItem myContextMenuItemB = new MenuItem(
        _resources, MENUITEM_TWO, 200000, 10) {
        public void run() {
            onMyMenuItemB();
        }
    };

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

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

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

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

public ContextMenuSample() {
    MainScreen mainScreen = new MainScreen();
    MyContextField myContextField = new MyContextField("Field label: ");
    mainScreen.add(myContextField);
    pushScreen(mainScreen);
}

Create a custom layout manager

Example: DiagonalManager.java

/**
 * DiagonalManager.java
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.custommenu;

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

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

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

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

    protected void sublayout(int width, int height) {
        int x = 0;
        int y = 0;
        Field field;
int numberOfFields = getFieldCount();

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

setExtent(width,height);

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

    if (index == this.getFieldWithFocusIndex()) {
        return super.nextFocus(direction, alt);
    } else {
        return index;
    }
}
Create a custom list

Example: SampleListFieldCallback.java

/**
 * SampleListFieldCallback.java
 *
 * Copyright (C) 2001-2005 Research In Motion Limited. All rights reserved.
 */

package com.rim.samples.docs.listfields;
import java.util.*;
import net.rim.device.api.system.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
public class SampleListFieldCallback extends UiApplication {
    private ListField myList;
    public static void main(String[] args) {
        SampleListFieldCallback app = new SampleListFieldCallback();
        app.enterEventDispatcher();
    }
    private static class ListCallback implements ListFieldCallback {
        private Vector listElements = new Vector();
        public void drawListRow(ListField list, Graphics g, int index, int y, int w) {
            String text = (String)listElements.elementAt(index);
            g.drawText(text, 0, y, 0, w);
        }
        public Object get(ListField list, int index) {
            return listElements.elementAt(index);
        }
    }
}
return listElements.elementAt(index);
}

public int indexOfList(ListField list, String p, int s) {
    return listElements.indexOf(p, s);
}

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

public void insert(String toInsert, int index) {
    listElements.addElement(toInsert);
}

public void erase() {
    listElements.removeAllElements();
}

public SampleListFieldCallback() {
    MainScreen mainScreen = new MainScreen();
    myList = new ListField();
    ListCallback myCallback = new ListCallback();
    myList.setCallback(myCallback);
    String fieldOne = "ListField one";
    String fieldTwo = "ListField two";
    String fieldThree = "ListField three";
    myList.insert(0);
    myCallback.insert(fieldOne, 0);
    myList.insert(1);
Create a menu item in a BlackBerry Java application

The menu item appears after a BlackBerry smartphone user views a contact in the address book. After a BlackBerry smartphone user clicks the menu item, the ContactsDemo application appears.

Example: DemoAppMenuItem.java

```java
/**
 * DemoAppMenuItem.java
 * Copyright (C) 2003-2007 Research In Motion Limited.
 *
 * The following code example creates a menu item that appears after
 * a user views a contact in the address book. After a user clicks the
 * menu item,
 * the Contacts Demo application appears.
 */

package com.rim.samples.docs.menuitem;

import net.rim.device.api.system.*;
import net.rim.device.api.ui.component.Dialog.*;
import net.rim.blackberry.api.menuitem.*;
import net.rim.blackberry.api.pdap.*;
import javax.microedition.pim.*;

public final class DemoAppMenuItem extends Application
```
private static final String ARG_LAUNCH_CONTACT_DEMO = "1";

public static void main(String[] args) {
    if (args == null || args.length == 0) {
        DemoAppMenuItem app = new DemoAppMenuItem();
        app.enterEventDispatcher();
    } else {
        String appToLaunch = args[0];
        if (ARG_LAUNCH_CONTACT_DEMO.equals(appToLaunch)) {
            new com.rim.samples.docs.contactsdemo.ContactsDemo().enterEventDispatcher();
        }
        // add more else ifs here
    }
}

DemoAppMenuItem() {
    long locationToAddMenuItem = ApplicationMenuItemRepository.MENUITEM_ADDRESSCARD_VIEW;
    addItem(ARG_LAUNCH_CONTACT_DEMO, locationToAddMenuItem, new ContactsDemoMenuItem());
    System.exit(0);
private static void addMenuItem(String argOfApp, long location, ApplicationMenuItem applMenuItem)
{
    ApplicationMenuItemRepository amir = ApplicationMenuItemRepository.getInstance();

    ApplicationDescriptor app = ApplicationDescriptor.currentApplicationDescriptor();

    //set the argument so that we know which app we want to have launched
    app = new ApplicationDescriptor(app, new String[]{ARG_LAUNCH_CONTACT_DEMO});

    amir.addMenuItem(location, applMenuItem, app);
}

/**
 * Create the menu item classes here
 */

private static class ContactsDemoMenuItem extends ApplicationMenuItem
{
    ContactsDemoMenuItem()
    {
        super(20);
    }

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

    public Object run(Object context)
    {
        BlackBerryContact c = (BlackBerryContact)context; //an error if this
        if ( c != null ) {
            Application.getApplication().requestForeground();
            Application.getApplication().requestForeground();
        }
    }
// on invocation, will call the main method of this app. with argument as specified in addMenuItem

} else {
    throw new IllegalStateException( "Context is null, expected a Contact instance");

}

return null;

}
Evaluating a mobile device UI

Review the following best practices to evaluate your UI:

- Did you use or extend existing UI components where possible so that your application can inherit the default behavior of the component?
- Did you follow the standard navigation model as closely as possible, so that a particular user action produces a consistent result across applications?
- Did you support and extend user tasks in useful ways? For example, after users download an application, does the application open automatically?
- Was the application saved in the Applications folder?
- Does the application focus on users’ immediate task?
- Does the application only display the information that a user needs at any one moment?
- Are the menu actions relevant to users’ current context?
- Can the number of times that users need to click the trackwheel, trackball, trackpad, or touch screen to complete a task be further minimized?
- Can users undo menu commands?
- Is the UI making effective use of the small screen?

Preverify a BlackBerry Java application

When you preverify your classes, you reduce the amount of processing that the BlackBerry smartphone must perform when you install your application. To partially verify your classes before you install your application on a BlackBerry smartphone, you can use the Preverify tool, available with the BlackBerry JDE. You can use the BlackBerry® Smartphone Simulator to preverify the .cod files.

Test applications on a BlackBerry Smartphone Simulator

After you develop and compile your application, you can test it on the BlackBerry smartphone. The most common first step is to set the BlackBerry JDE to use a BlackBerry Smartphone Simulator. The BlackBerry Smartphone Simulator runs the same Java code as the BlackBerry smartphones, so the BlackBerry Smartphone Simulator provides an accurate environment for testing how applications will function on a BlackBerry smartphone. The BlackBerry JDE includes current versions of the BlackBerry Smartphone Simulator.
**BlackBerry Smartphone Simulator**

The BlackBerry Smartphone Simulator is designed to emulate a BlackBerry experience without using a real BlackBerry smartphone.

The BlackBerry Smartphone Simulator is an application that you install on your computer that shows an image of the BlackBerry smartphone model of your choice. This image has the look and feel of an actual BlackBerry smartphone. The functionality includes the same user interaction of an actual BlackBerry smartphone (including the trackwheel or trackball and the keyboard), the same applications, and the same features, such as email messages, phone, and Internet browsing.

The BlackBerry Smartphone Simulator also serves as a platform on which applications can run. This includes the ability for the applications to make network connections, store data, and handle email messages. The BlackBerry Smartphone Simulator includes the BlackBerry smartphone applications that are typically available on BlackBerry smartphones and you can install and test your own applications. You can simulate and test various connectivity and state changes using the BlackBerry Smartphone Simulator.

When you use the BlackBerry Smartphone Simulator to perform testing, you can simulate additional BlackBerry services.

The BlackBerry® MDS Simulator and the BlackBerry® Email Simulator are available for this purpose.

To get the BlackBerry Smartphone Simulator, visit www.blackberry.com/developers and download the BlackBerry JDE or the BlackBerry JDE Component Package.

**BlackBerry MDS Simulator**

The BlackBerry MDS Simulator is designed to simulate the BlackBerry MDS Connection Service component of the BlackBerry® Enterprise Server. When you use the BlackBerry Smartphone Simulator with the BlackBerry MDS Simulator you can test network, push HTTP, and browser applications that are designed for use with a BlackBerry Enterprise Server. To get the BlackBerry MDS Simulator, visit www.blackberry.com/developers and download the BlackBerry Email and MDS Services Simulator Package.

**BlackBerry Email Simulator**

The BlackBerry Email Simulator is designed to send and receive messages between the BlackBerry Smartphone Simulator and either a messaging application, such as Microsoft® Outlook®, or POP3 and SMTP servers. You do not require a BlackBerry Enterprise Server. To get the BlackBerry Email Simulator, visit www.blackberry.com/developers and download the BlackBerry Email and MDS Services Simulator Package.
Test applications on a BlackBerry smartphone

After you test your application on the BlackBerry Smartphone Simulator, you can install your application on a BlackBerry smartphone.

If your application uses signed APIs, you can need code signing keys. After you install the application on the BlackBerry smartphone, you can open the application and test its functionality and performance.

For debugging purposes, you can attach your BlackBerry smartphone 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.
1. List three steps to evaluate a mobile device UI.

2. True or false. The BlackBerry Smartphone Simulator runs the same Java code as the BlackBerry smartphones.

3. Which simulator can you use to test network, push HTTP, and browser applications that are designed for use with a BlackBerry Enterprise Server?
Answers

1. You can use the following three steps to evaluate a mobile device UI:
   - Review best practices to evaluate your UI
   - Preverify a BlackBerry Java application
   - Test applications on a BlackBerry smartphone

2. True

3. BlackBerry MDS Simulator
Developing a user interface for mobile devices

The Java ME MIDP standard addresses the API and BlackBerry JVM needs of a constrained wireless device with a UI. Developers can often build one Java application using the MIDP standard APIs and run that application on many different types of devices.

A CLDC application extends the UiApplication class and starts with a standard main() method. BlackBerry UI APIs provide more functionality and flexibility than the standard javax.microedition.lcduilib library. However, applications are not portable to other devices.

Applications designed for BlackBerry smartphones can provide a balance between the best possible user experience and a long battery life.

The BlackBerry UI APIs are a library of UI components that are designed to provide default layouts and behaviors that are consistent with the core BlackBerry smartphone applications.

- Field components provide standard UI elements for date selection, options, check boxes, lists, text fields and labels, and progress bar controls.
- Manager components provide an application with the ability to arrange components on a BlackBerry smartphone screen in standard ways, such as horizontally, vertically, or in a left-to-right flow.
- Screen components provide a standard screen layout, a default menu, and standard behavior when the BlackBerry smartphone user presses the Escape key or clicks the trackwheel or trackball.
- Graphics components represent the entire drawing surface that is available to the BlackBerry Java application.

After you create a screen, you can add fields and a menu to the screen and display it to the BlackBerry smartphone user by pushing it on to the UI stack. The Menu object has associated menu items that are runnable objects, which perform a specific task when the BlackBerry smartphone user selects one of the items. For more sophisticated custom BlackBerry Java applications, you can customize the BlackBerry smartphone UI and implement field types, as required. You can also add custom navigation and trackwheel behavior.

Be sure to evaluate your UI. Review best practices for a mobile device UI. Preverify your BlackBerry Java application. Test your application on a BlackBerry Smartphone Simulator. Finally, test your applications on a BlackBerry smartphone.

Summary

The Java ME MIDP standard addresses the API and BlackBerry JVM needs of a constrained wireless device with a UI. Developers can often build one Java application using the MIDP standard APIs and run that application on many different types of devices.

A CLDC application extends the UiApplication class and starts with a standard main() method. BlackBerry UI APIs provide more functionality and flexibility than the standard javax.microedition.lcduilib library. However, applications are not portable to other devices.

Applications designed for BlackBerry smartphones can provide a balance between the best possible user experience and a long battery life.

The BlackBerry UI APIs are a library of UI components that are designed to provide default layouts and behaviors that are consistent with the core BlackBerry smartphone applications.

- Field components provide standard UI elements for date selection, options, check boxes, lists, text fields and labels, and progress bar controls.
- Manager components provide an application with the ability to arrange components on a BlackBerry smartphone screen in standard ways, such as horizontally, vertically, or in a left-to-right flow.
- Screen components provide a standard screen layout, a default menu, and standard behavior when the BlackBerry smartphone user presses the Escape key or clicks the trackwheel or trackball.
- Graphics components represent the entire drawing surface that is available to the BlackBerry Java application.

After you create a screen, you can add fields and a menu to the screen and display it to the BlackBerry smartphone user by pushing it on to the UI stack. The Menu object has associated menu items that are runnable objects, which perform a specific task when the BlackBerry smartphone user selects one of the items. For more sophisticated custom BlackBerry Java applications, you can customize the BlackBerry smartphone UI and implement field types, as required. You can also add custom navigation and trackwheel behavior.

Be sure to evaluate your UI. Review best practices for a mobile device UI. Preverify your BlackBerry Java application. Test your application on a BlackBerry Smartphone Simulator. Finally, test your applications on a BlackBerry smartphone.
1. Which of the following statements are false?

A. Only one screen in a BlackBerry Java application is active at one time.
B. The main structure for a BlackBerry smartphone UI is the Screen object.
C. You can only display one screen at a time in a BlackBerry Java application.
D. After you create a screen, you can add fields and a menu to the screen and display it to the BlackBerry smartphone user by pushing it on to the UI stack.

2. Which of the following apply to the MIDP standard?

A. One disadvantage to using the MIDP standard is that applications are not portable to other smartphones.
B. The MIDP standard provides a core set of Java APIs that the BlackBerry smartphone can support, regardless of its underlying operating system.
C. All of the core BlackBerry applications are built using only the Java ME MIDP standard as defined in JSR 118.
D. The Java ME MIDP standard addresses the API and BlackBerry JVM needs of a constrained wireless smartphone with a UI.

3. A CLDC application extends the _____ class and starts with a _____ method.

A. UiApplication    standard main()
B. Screen           navigation()
C. Field            run()
D. Manager          screen()

4. What are the advantages to using the BlackBerry CLDC application?

A. Applications can reuse library modules that CLDC applications can import.
B. Applications can start automatically in the background after the smartphone turns on.
C. Applications must run active background threads after they have closed.
D. Applications are portable to other smartphones.

5. Which of the following is not a best practice for designing applications for BlackBerry smartphones?
   A. Design your UI to allow users to undo commands.
   B. Follow the standard navigation model as closely as possible.
   C. Encourage users to simultaneously display many screens.
   D. Support and extend user tasks in useful ways.

6. List the components of the BlackBerry smartphone UI.

7. What is the main structure for a BlackBerry smartphone UI?

8. List three methods to evaluate a mobile device UI.