



CMER

Centre for Mobile Education and Research

GUI Components Part I



Overview

- **UiApplication Class**
- **MainScreen Class**
- **BitmapField Class**
- **ButtonField Class**
- **CheckboxField Class**
- **Dialog Class**
- **EditField Class**
- **GuageField Class**
- **LabelField Class**
- **Menu Class**
- **RadioButtonField Class**
- **RadioButtonGroup**
- **TextField Class**



UIApplication Class

- *public class HelloWorld extends UIApplication*
- Base class for all device applications that provide a user interface.
- A UI application maintains a stack of Screen objects.
- As it pushes screens onto the stack, it draws them on top of any other screens already on the stack.
- When the application pops a screen off the stack, it redraws the underlying screens as necessary.
- Only the screen on the top of the stack receives input events.



UIApplication Class (Cont.)

- Each screen may appear only once in the display stack.
- The application throws a runtime exception if you attempt to push a single screen onto the stack more than once.
- A UI application must follow rules similar to those of traditional Swing applications.
 - Only one thread at a time (generally, the event-dispatching thread) may gain access to an interface component at once.
 - If you need access to the UI from outside event-handling or drawing code, then you can use the `invokeLater()` or `invokeAndWait()` methods.



UiApplication Class (Cont.)

- `pushScreen(new MScreen());`
 - This will push the `MScreen` instance onto the UI stack to be displayed.
- For more information on `UiApplication`, refer to the BlackBerry JDE API Specification.
 - `net.rim.device.api.ui.UiApplication`



MainScreen Class

- *class MScreen extends MainScreen*
- A full screen providing features common to standard RIM device applications.
- MainScreen objects contain a title section, a separator element, and a main scrollable section.
- For more information on MainScreen, refer to the BlackBerry JDE API Specification.
 - *net.rim.device.api.ui.container.MainScreen*
 - Extends *net.rim.device.api.ui.container.FullScreen*



net.rim.device.api.ui.component

- Provides a library of prebuilt interface components and controls for constructing usable UI applications
 - textfields
 - labels
 - buttons
 - etc..
- For more information on this library and a list of its ui components, refer to the BlackBerry JDE API Specification.
 - net.rim.device.api.ui.component



BitmapField Class

- Displays a bitmap.
- The BlackBerry JDE supports GIF, JPG, and PNG image formats;
- PNG files are the most efficient file format for BlackBerry devices with limited memory.
- By default this field uses only enough area to fit its contained bitmap.
- If you specify an alignment style at construction, the field always uses the full width of the controlling manager's layout.
- By specifying the Field.FOCUSABLE style at construction, you can make this field focusable.



BitmapField (Cont.)

- To add a graphic to your application, you must first add the graphic file to your project.
- In your application, you can refer to your graphic by its name, by using `Bitmap.getBitmapResource()`.
- After you have the bitmap, you can add it to a bitmap field and then add that field to the screen.



BitmapField Example

```
Bitmap b = new Bitmap(200, 40);  
b = Bitmap.getBitmapResource("bb.gif");  
BitmapField bf = new BitmapField(b);  
bf.setSpace(Graphics.getScreenWidth()/2 - b.getWidth()/2,  
            Graphics.getScreenHeight()/2 - b.getHeight()/2 );  
add(bf);
```



BitmapField Example Explained

- Assumption is made that a GIF image named *bb.gif* has previously been added to the project.
- First, a bitmap of size 200 x 40 is created, which mimics the actual size of *bb.gif*.
- Then a BitmapField is created from that bitmap image
- The BitmapField is then centered using *setSpace*, and added finally to the screen.



ButtonField Class

- Displays a button.
- Contains a button control.
- When the user presses a ButtonField the FieldChangeListener that was set by invoking `setChangeListener()` is notified by its `fieldChanged()` method being invoked.
- Refer to the BlackBerry JDE API Specification for more information and how this is done.
 - `net.rim.device.api.ui.component.ButtonField`



ButtonField Example

```
ButtonField btn = new ButtonField("myButton");  
btn.setChangeListener(new ButtonListener()); add(btn);
```

```
final class ButtonListener implements FieldChangeListener  
{  
    public void fieldChanged(Field field, int context) {  
        ButtonField btn = (ButtonField) field;  
        Status.show("Button pressed: " + btn.getLabel());  
    }  
}
```



ButtonField Example Explained

- The previous code sample demonstrates how to create a simple ButtonField and attach a listener to it.
- First a button called “myButton” is created.
- Next, a custom listener is added to the button.
- This listener would “listen” to this button’s events, if it has been clicked it will display the message “Button pressed: myButton” in the status field.
- Finally, the button is added on to the screen.



CheckboxField Class

- Displays a checkbox before the text label.
- In the default font, this box appears either empty, or containing a check mark, depending upon the state of the field.
- When this field has focus, the box appears inverted.
- Pressing the spacebar when a checkbox field has the focus toggles its state.



CheckboxField Example

```
CheckboxField chk = new CheckboxField("Checkbox 1",  
    true);  
CheckboxField chk2 = new CheckboxField("Checkbox 2",  
    false);  
add(chk);  
add(chk2);
```

- The above sample code creates two checkboxes.
- The first checkbox is selected by default while the second is not.



Dialog Class

- Provides a dialog box with predefined configurations.
- To get a standard, predefined dialog use:
 - `alert(java.lang.String)`
 - `ask(int)`
 - `inform(java.lang.String)`
- These pop up a predefined dialog and wait for user input.
- To get a more customized dialog, instantiate this class or extend it.
- Pressing ESCAPE returns CANCEL, but only if it was one of the choices specified in the values array.



Dialog Examples

- `Dialog.alert("Bye World!");`
 - Displays an alert dialog box that shows the message “Bye World!”
- `Dialog.ask(Dialog.D_YES_NO, “Exit?”);`
 - Prompts the user with the message “Exit?” and the options of either “Yes” or “No”
- `Dialog.inform(“Connection lost.”);`
 - Displays a notification dialog with the message “Connection lost.”



EditField Class

- An editable simple text field with no formatting.
- Displays a label in front of the text contents.
- If the contents occupy more than one line the text will flow around the label.
- The caret position behaves as if it were between characters in the text contents, however it appears as a full inverted box on the character immediately following the caret.
- If this field is `Field.EDITABLE`, typing inserts text into the contents at the caret position.
- The `BACKSPACE` character removes the character prior to the caret, and the `DELETE` character removes the character after the caret.
- Example:

```
EditField edit = new EditField("Username: ", "");  
add(edit);
```



GaugeField Class

- Displays a label followed by a gauge bar.
- A GaugeField object can be used to display a progress bar while the application processes information.
- The gauge optionally has text overlaid over it indicating the percentage the gauge represents.
- If this field is built as Field.EDITABLE, the user can employ the trackwheel to change the value by ALT-rolling.
- If Ui.getIncreaseDirection() returns -1, then rolling the trackwheel down decreases the value.
- If Ui.getIncreaseDirection() returns +1, then rolling the trackwheel up increases the * value.



GaugeField Example

```
MainScreen mainScreen = new  
    MainScreen(Field.USE_ALL_HEIGHT|Field.FIELD_LEFT);  
GaugeField gaugeField = new GaugeField ("PROGRESS ", 0, 30,  
    15, Field.FOCUSABLE);  
gaugeField.setEditable(true);  
mainScreen.add (gaugeField);
```

```
GaugeField gaugeField2 = new GaugeField ("PERCENT ", 0, 30,  
    15, GaugeField.PERCENT);  
mainScreen.add (gaugeField2);
```

```
GaugeField gaugeField3 = new GaugeField ("NO_TEXT ", 0, 30,  
    15, GaugeField.NO_TEXT);  
mainScreen.add (gaugeField3);  
pushScreen (mainScreen);
```



GaugeField Example Explained

- The previous code sample creates three different progress bars
- the first GaugeField is the default progress bar. It is set as editable so that the user can change the progress setting.
- The second progress bar shows the progress in terms of percentages.
- The last progress bar does not display any text in the progress bar and displays a blank gauge.
- All three progress bars have a minimum value of 0 and maximum value of 30.



LabelField Class

- **Displays a label.**
- **Optionally focusable.**
- **Optionally can shorten text (with an ellipsis) that is too long.**



LabelField Example

```
add(new LabelField("LabelField"));
```

```
add(new LabelField("LabelField 2", 0, -1,  
    Field.FIELD_RIGHT));
```

```
LabelField lbl = new LabelField("LabelField 3", 0, -1,  
    Field.FIELD_HCENTER);
```

```
Font fnt = this.getFont().derive(Font.BOLD |  
    Font.ITALIC);
```

```
lbl.setFont(fnt);
```

```
add(lbl);
```




LabelField Example Explained

- The previous code sample creates three different LabelFields
- The first LabelField is the default label. It is horizontally-aligned to the left and uses the default font.
- The second LabelField is horizontally-aligned to the right, again with the default font.
- The last LabelField is horizontally-centered on the screen with italicized fonts.



ListField Class

- Contains rows of selectable list items.
- A ListField uses a class that implements the ListFieldCallback interface to perform drawing tasks.
- A ListField must register a class that implements the ListFieldCallback interface using the setCallback method before the class can be used.
- After registration, when a ListField must display an item in its list, it invokes the appropriate methods of the registered callback class.



Menu Class

- Displays a menu in the top right corner of the screen.
- A menu is a vertically arranged list of items.
- The currently selected menu item is marked by inverting its rectangle.
- To choose the selected menu item, one can either click the trackwheel or press ENTER.
- To dismiss the menu without choosing an item, one can press ESCAPE.
- Menus also support prefix searching; if one presses a letter key, the next menu item that starts with that character gets selected.
- A SeparatorField object can be added to a menu in the following ways:
 - use the add(ContextMenu, boolean) method and set the addSeparator argument to true.
 - use the addSeparator() method to add a separator.



PasswordEditField Class

- This field stores the password as plain text but draws it as a series of asterisks, one for each character cluster (group of characters treated as one after diacritics and ligatures have been handled).
- This field does not have any automatic input replacement, so the typed text is exactly what is typed.
- AutoCaps, AutoText, AutoPeriod, and any other transformation are turned off.
- Holding a key and rolling the wheel is not allowed.
- This field does not support copy or cut operations.
- By default, this field supports all the available characters.
- You can restrict this by applying a TextFilter.
- Example:

```
PasswordEditField pass = new PasswordEditField("Password: ", "");  
add(pass);
```



RadioButtonField Class

- Displays a circular button to the left of the text label.
- With the default font, this button is either empty or contain a solid circle depending on the state of the field.
- When this field has the focus, the button appears inverted.
- You group together Radio buttons fields using the RadioButtonGroup class.
- At most one button within the group can be selected at any one time.
- The group can be programmatically created or changed so that no button is selected, but a user cannot deselect all fields in the group.
- If the control is EDITABLE (radio button fields are created this way by default), pressing ENTER selects this field and deselects the previously selected radio button field in the group.



RadioButtonGroup Class

- Groups a set of related radio button fields.
- The index of a RadioButtonField in the group does not necessarily correspond to the order of the buttons on screen.
- Indices are assigned in the order that buttons are added to the group.
- Each Radio button field can belong to only one group at a time.



RadioButton Example

```
RadioButtonGroup rbg = new RadioButtonGroup();  
RadioButtonField rbf1 = new RadioButtonField("Radio 1",  
    rbg, false);  
RadioButtonField rbf2 = new RadioButtonField("Radio 2",  
    rbg, true);  
RadioButtonField rbf3 = new RadioButtonField("Radio 3",  
    rbg, false);  
add(rbf1);  
add(rbf2);  
add(rbf3);
```



RadioButton

Example Explained

- The *RadioButtonField* works with the *RadioButtonGroup* to function just as any radio button does on a web form.
- First the *RadioButtonGroup* is created.
- Then each *RadioButtonField* is created while assigning it to the group.
- Finally, each radio button is added to the screen (not the button group).
- Notice that the label for the radio button and the initial status (selected/unselected) with a true or false value are provided.



RichTextField Class

- A readonly text field for showing text in a variety of fonts and formatting.
- When building a rich text field that contains formatted text, you specify a list of offsets.
- These offsets define the boundaries of the various formatting regions for the field's text.
 - The first offset position in the list marks the beginning of the field's text (thus is zero), and the last offset position in the list marks the end of the field's text (this is equal to the field's text length).
- The field also has an attribute list, corresponding to the regions described in the offset list.
- Each element's value in the attribute list is an index into the fonts list.



RichTextField Class (Cont.)

- The fonts list contains the various formatting styles used by the field's contained text.
- Any font element with a null value that's used by the attribute list simply indicates that the field should draw the text region using the default font.
- You can also build a RichTextField with a list of cookie objects corresponding to various regions, as determined by the offsets list.
- You can use the Field.NON_FOCUSABLE style to build a rich text field that cannot accept the focus.
- See the BlackBerry Application Developer Guide Volume 1 for more information on formatting text in a RichTextField.



RichTextField Example

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

```
String str[] = new String[] {"RichTextField:", "Value"};
```

```
int off[] = new int[] {0, str[0].length(), str[0].length() +  
    str[1].length()};
```

```
byte attr[] = new byte[] {0, 1};
```

```
FontFamily fontfam[] = FontFamily.getFontFamilies();
```

```
Font fon[] = new Font[2]; fon[0] =
```

```
    fontfam[0].getFont(FontFamily.SCALABLE_FONT, 16);
```

```
fon[1] = fontfam[1].getFont(FontFamily.SCALABLE_FONT, 18);
```

```
add(new RichTextField(str[0] + str[1], off, attr, fon,  
    RichTextField.TEXT_ALIGN_HCENTER));
```



RichTextField Example Explained

- The previous code sample creates two different RichTextFields.
- The first RichTextField is the default. It is horizontally-aligned to the left and uses the default font.
- The second RichTextField is created by specifying a list of texts to be displayed, their appropriate offsets, list of fonts to be used, and is horizontally-centered.



SeparatorField Class

- A field which draws a horizontal line across its width.
- Displays a separator, currently always a horizontal line.
- A SeparatorField object can be added to a menu in the following ways:
 - use the `Menu.add(ContextMenu, boolean)` method and set the `addSeparator` argument to true.
 - use the `Menu.addSeparator()` method to add a separator.
- Example:
`add(new SeparatorField());`



Status Class

- Simple dialog to show ongoing status.
- This extension of `PopupScreen` lets you show a simple status screen which the user can dismiss with a trackwheel click or by typing `SPACE` or `ESCAPE`.
- Example:
`Status.show("This is the current status.");`



TextField Class

- Displays a label in front of the text contents.
- If the contents occupy more than one line the text will flow around the label.
- The caret position behaves as if it were between characters in the text contents, however it appears as a full inverted box on the character immediately following the caret.
- The trackwheel moves the focus caret in the major direction (usually vertical), and using alt+trackwheel will move the focus caret in a less significant direction (usually horizontal).
- If this field is Field.EDITABLE, typing inserts text into the contents at the caret position.
- The BACKSPACE character removes the character prior to the caret, and the DELETE character removes the character after the caret.
- Some subclasses of this may choose to support special symbols:
 - typing a Sym brings up the symbol screen from which the user can select a symbol.
 - Depending on the keyboard Sym could alternatively be Alt+Space or Alt+ZX.