BlackBerry Java Application Accessibility

Version: 5.0

Development Guide

BlackBerry

Published: 2010-05-04
SWD-811646-0504085720-001

Contents

Understanding accessibility

1

Accessibility refers to the extent to which a product or service can be used by as many people as possible. In this context, accessibility means designing applications so that people with disabilities or impairments can use the applications on a BlackBerry® device.

When you design your BlackBerry device application, consider the following users:

- blind people
- visually impaired people
- colorblind people
- deaf people
- hearing impaired people
- speech impaired people
- people with motor impairments
- people with cognitive or learning disabilities

Like any other group of users, people with disabilities or impairments have needs, wants, and expectations about the behavior of applications. Some of the reasons to make your applications accessible to as many users as possible include the following possible benefits:

- Social responsibility: Making it easier for people with disabilities or impairments to benefit from the functionality that your application provides can promote equality.
- Market share: Ensuring that your application can be used by people with disabilities or impairments increases the number of people who can purchase and benefit from your application.
- Compliance: Addressing the applicable guidelines and regulatory requirements (such as Section 508 of the Rehabilitation Act in the United States) that dictate that your application must be accessible by people with disabilities or impairments can allow you to enter certain markets.

Accessible applications

An accessible application is one that can be used effectively by people with disabilities or impairments. Making your application accessible involves the following activities:

- Designing your application's UI with all of your users in mind, including people with disabilities or impairments.
- Providing information about your application to assistive technologies such as screen readers. For example, an accessible application might provide information about a text field that is displayed on a screen to a screen reader. The screen reader can convert the text to speech for people who are blind or vision-impaired.

RIM provides best practices to follow when designing an accessible application's UI, and RIM provides the Accessibility API that your application can use to provide information to assistive technologies.

Related topics
Best practice: Designing accessible applications, 4
Developing accessible BlackBerry device applications by using the Accessibility API, 5

Assistive technology

An assistive technology device or assistive technology application can respond to the information that it receives from an accessible application and can render the application usable by people with disabilities or impairments. One example of an assistive technology application is a screen reader. A screen reader can receive information about the items that the BlackBerry® device screen displays and the actions that occur. The screen reader can re-present the display or action to the user, such as through audio output or braille.

Some examples of assistive technology devices are screen magnifiers and keyboards designed for use by people with motor impairments.

RIM provides the Accessibility API that you can use in your assistive technology applications to receive information from accessible applications.

Related topics
Enable an assistive technology application to receive notification of UI events, 18

Support for accessibility on BlackBerry devices

A BlackBerry® device can include the following features to facilitate its use by people with disabilities or impairments:

- speakerphone
- visual, audible, and vibration alerts and notifications
- hearing aid compatibility (on some models)
- assignable ring tones to identify callers
- customizable fonts and themes
- browser zoom
- reverse contrast and grayscale display settings
- autoText
- SureType® technology with predictive text
- SurePress™ touch screen

For more information about accessibility on BlackBerry devices, visit www.blackberry.com/accessibility.

Best practice: Designing accessible applications 2

Guidelines for UI design

- Stay focused on users' immediate task. Display only the information that users need at any one moment. For example, simplify data selection and presentation by displaying information in a logical order.
- Group controls according to common usage or common functionality to minimize the cognitive load for users.
- Provide enough space between controls so that users can distinguish one control from another.
- Use UI components consistently so that users can recognize common UI components easily. For example, use buttons to initiate actions. Avoid using other controls, such as hyperlinks, to initiate actions.
- If you are designing an application that supports an assistive technology device, such as a screen reader, and you do not use BlackBerry® UI APIs or support the Accessibility API, expose the unique UI components in your application programmatically so that assistive technology can interpret the information.

Guidelines for navigation

- When application screens open, set the focus to the control that users are most likely to use first. This approach simplifies tasks for users.
- Indicate clearly the UI component that has focus. For example, use white text on a blue background.
- Where possible, allow users to use the keyboard to initiate the most frequently used actions in the application. For example, allow users to press the Enter key to select a menu item.
- Where possible, inform users of important events, such as calendar reminders, in multiple ways. For example, provide a sound effect and a visual notification for the reminder.
- Where possible, apply redundancy to provide users with multiple ways to interact with common actions. For example, use the Menu key to allow users to access the full menu and the trackball or trackpad to allow users to access the context menu. On BlackBerry devices with a touch screen, in some cases, users can click the screen to access the context menu.
- In each menu, set the default menu item as the item that users are most likely to select. The default item in the context menu should be the same as the default item in the full menu.
- If a process or application requires users to complete a series of lengthy or complex steps, list all the steps or screens where possible. Identify the steps that are complete, in progress, and not yet started. For example, include a table of contents in wizards. If users close a wizard, they can use the table of contents to return to a specific location in the wizard.

Guidelines for text

- Provide specific messages. To support error recovery, use one short sentence that states clearly the reason for displaying the message and the actions that can dismiss it.
- Where possible, inherit the font settings that the user has set.

Guidelines for color and images

- Avoid using color as the only means of communication. For example, instead of using only red text to notify users of a critical action, consider placing a red symbol, such as a red exclamation mark, beside the text instead.
- Choose colors that have high contrast, such as black, white, navy blue, and yellow.
- To help users to distinguish between adjacent UI components (such as alternating messages in an SMS text message thread) and to distinguish between background and foreground colors, use colors that result in a contrast ratio of 7:1 or higher.
- Add contextual information to images, such as the image name, to communicate the meaning and context of the images.

Developing accessible BlackBerry device applications by using the Accessibility API

3

Accessibility API concepts

Before you use the Accessibility API, you should understand the following concepts.

Concept	Description
role	Each accessible UI object has a role. A role specifies the type of UI component. Roles include text fields, labels, screens, dialog boxes, lists, icons, radio buttons, tables, check boxes, and buttons. The `AccessibleRole` interface in the Accessibility API defines roles as constants.
state	Each accessible UI object has a state. A state specifies the current object state as a mask of flags. States include focusable, focused, checked, active, selected, and editable. Objects can have multiple states at a time, such as focusable and focused. The `AccessibleState` interface defines states as constants.
container	Some accessible UI objects are containers. Containers are visual elements, such as screens, dialog boxes, lists, and menus, that contain children. You can discover the children of a container with methods in the `AccessibleContext` interface.
table	A table is a type of container that displays children in a tabular format. Examples of tables include the monthly view and weekly view in the calendar, and a table on a web page. You can discover the table properties for an accessible UI object by using the methods in the `AccessibleTable` interface.

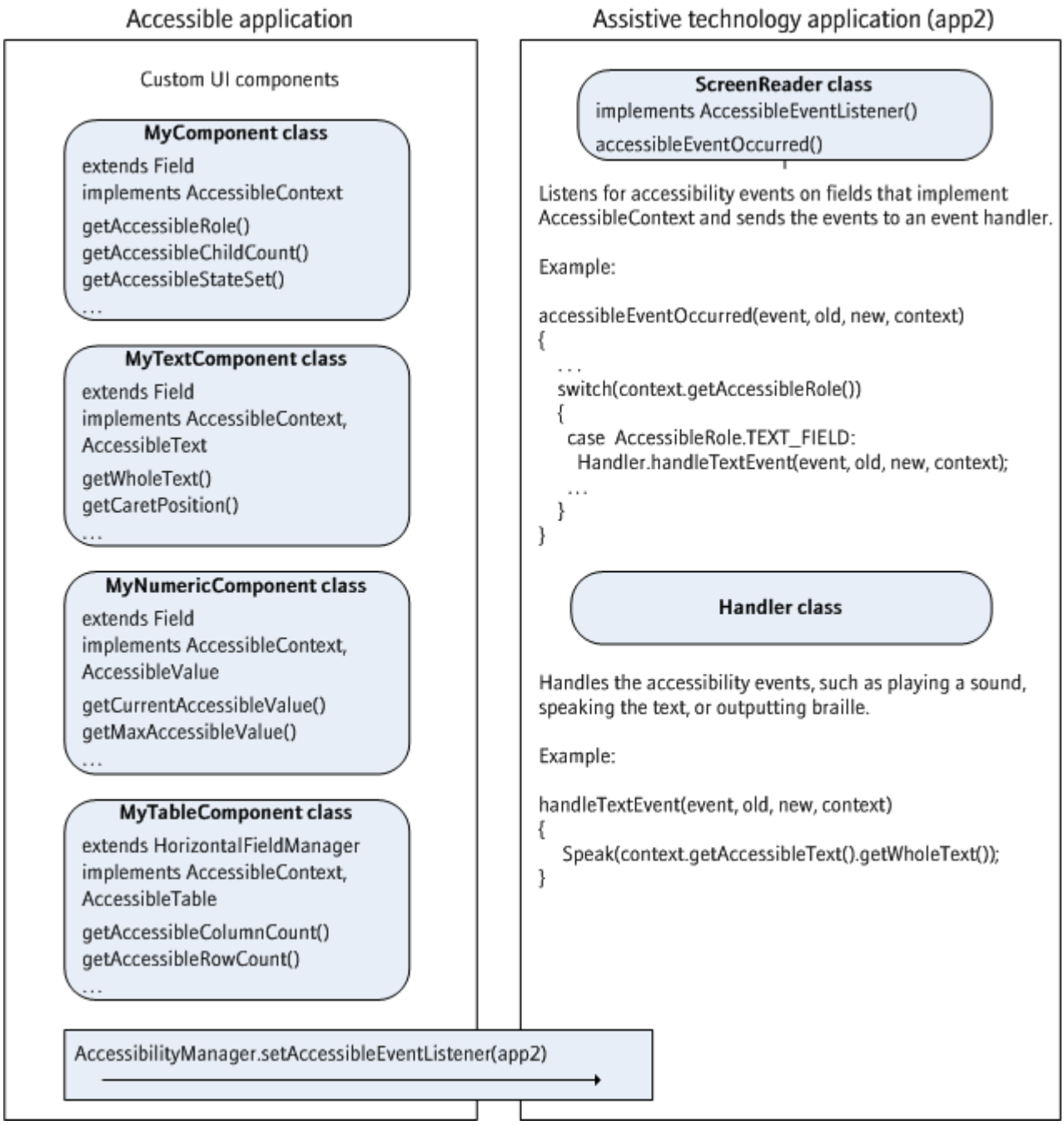
Introduction to the Accessibility API

The Accessibility API enables you to develop accessible BlackBerry® device applications that provide information to assistive technology applications such as screen readers. Screen readers and other assistive technologies can make it easier for people with disabilities or impairments to use your accessible application. Standard UI components, such as `TextField`, automatically provide information to assistive technology applications. If you use custom UI components (components that extend the standard UI components), you must use the Accessibility API to provide information to assistive technology applications.

For all the custom UI components in your accessible application, you must implement the `net.rim.device.api.ui.accessibility.AccessibleContext` interface. The methods in this interface identify the information that you want to send to an assistive technology application. For example, in a custom UI component that implements `AccessibleContext`, `getAccessibleRole()` returns the control's role, such as `AccessibleRole.TEXT_FIELD`. If your application contains custom text fields, custom numeric fields, or custom tables, you can also use the `AccessibleText`, `AccessibleValue`, or `AccessibleTable` interfaces.

An assistive technology application registers as an accessibility event listener. When an accessibility event occurs on a custom UI component, the accessible application invokes the `accessibleEventOccurred()` method in the registered assistive technology application and passes in the event information. The assistive technology application handles the event and performs the appropriate action, such as speaking the text or playing a sound.

The following diagram shows the relationship between an accessible application and an assistive technology application. The AccessibilityDemo sample application that is provided with BlackBerry® Java® Development Environment and BlackBerry® Java® Plug-in for Eclipse® 4.6.1 and later is structured this way.


Accessible application
Custom UI components
MyComponent class
extends Field
implements AccessibleContext
getAccessibleRole()
getAccessibleChildCount()
getAccessibleStateSet()
...
MyTextComponent class
extends Field
implements AccessibleContext, AccessibleText
getWholeText()
getCaretPosition()
...
MyNumericComponent class
extends Field
implements AccessibleContext, AccessibleValue
getCurrentAccessibleValue()
getMaxAccessibleValue()
...
MyTableComponent class
extends HorizontalFieldManager
implements AccessibleContext, AccessibleTable
getAccessibleColumnCount()
getAccessibleRowCount()
...
AccessibilityManager.setAccessibleEventListener(app2)
Assistive technology application (app2)
ScreenReader class
implements AccessibleEventListener()
accessibleEventOccurred()
Listens for accessibility events on fields that implement AccessibleContext and sends the events to an event handler.
Example:
accessibleEventOccurred(event, old, new, context)
{
...
switch(context.getAccessibleRole())
{
case AccessibleRole.TEXT_FIELD:
Handler.handleTextEvent(event, old, new, context);
...
}
}
Handler class
Handles the accessibility events, such as playing a sound, speaking the text, or outputting braille.
Example:
handleTextEvent(event, old, new, context)
{
Speak(context.getAccessibleText().getWholeText());
}

Related topics
The AccessibilityDemo sample application, 8

The AccessibilityDemo sample application

BlackBerry® Java® Development Environment 4.6.1 and later and BlackBerry® Java® Plug-in for Eclipse® with BlackBerry Component Package 4.6.1 and later provide a sample application that demonstrates the communication between an accessible application and an assistive technology application. The sample application can also be downloaded from http://www.blackberry.com/go/accessibilitysample.

The AccessibilityDemo application consists of two projects.

Project	Description
CustomComponentsDemo	CustomComponentsDemo is the accessible application. The screen contains custom UI components that implement the `AccessibleContext` interface, and, if appropriate, the `AccessibleText` interface, the `AccessibleTable` interface, or the `AccessibleValue` interface. The `AccessibleContext` interface provides information about the accessible object, such as its value and state, to the assistive technology application. The UI components broadcast accessibility events, such as when focus is set on a component or when a component's value changes.
ScreenReaderDemo	ScreenReaderDemo is the assistive technology application, a screen reader. The `ScreenReader` class is registered as an accessibility event listener and implements the `AccessibleEventListener` interface. `ScreenReader` receives the accessibility events that CustomComponentsDemo broadcasts. For demonstration purposes, the screen reader outputs `System.out.println()` statements to handle the accessibility events.

Related topics
Explore the AccessibilityDemo sample application, 10
Set up the AccessibilityDemo sample application in the BlackBerry Java Plug-in for Eclipse, 8
Set up the AccessibilityDemo sample application in the BlackBerry Java Development Environment, 9

Set up the AccessibilityDemo sample application in the BlackBerry Java Plug-in for Eclipse

Before you begin: Verify that you are using BlackBerry® Java® Plug-in for Eclipse® 1.1 or later with BlackBerry JDE Component Package 4.6.1 or later.

1. In Eclipse, open a workspace.
2. On the **File** menu, click **Import**.
3. In the **Import** dialog box, expand the **BlackBerry** folder and select the **Import BlackBerry Samples** option.
4. Click **Next**.
5. In the **Import BlackBerry Samples** dialog box, click **Deselect All**.
6. Select the check box beside the **CustomComponentsDemo** project.
   The check box beside the **ScreenReaderDemo** project is automatically selected.
7. Click **Finish**.

The CustomComponentsDemo and ScreenReaderDemo projects display in the Package Explorer. The BlackBerry Java Plug-in for Eclipse sets the Java build path for CustomComponentsDemo to include ScreenReaderDemo, which CustomComponentsDemo references.

Related topics
The AccessibilityDemo sample application, 8
Set up the AccessibilityDemo sample application in the BlackBerry Java Development Environment, 9

Set up the AccessibilityDemo sample application in the BlackBerry Java Development Environment

Before you begin: Verify that you are using BlackBerry® Java® Development Environment 4.6.1 or later.

1. In the BlackBerry JDE, open the **samples** workspace.
   a. On the **File** menu, click **Open Workspace**.
   b. Navigate to the **samples** folder.
   c. Select the **samples.jdw** file.
   d. Click **Open**.
2. Expand the **accessibilitydemo** folder.
3. Right-click the **CustomComponentsDemo** project and click **Activate Project**.
4. Right-click the **ScreenReaderDemo** project and click **Activate Project**.

Related topics
The AccessibilityDemo sample application, 8
Set up the AccessibilityDemo sample application in the BlackBerry Java Plug-in for Eclipse, 8

Explore the AccessibilityDemo sample application

You can explore the AccessibilityDemo sample application to help increase your understanding of the communication between an accessible application and an assistive technology application.

Before you begin: Set up the sample application in either the BlackBerry® Java® Plug-in for Eclipse® or the BlackBerry® Java® Development Environment.

1. Perform one of the following actions to display the console window:
   - In the BlackBerry Java Plug-in for Eclipse, on the **Window** menu, select **Show View** > **Console**.
   - In the BlackBerry JDE, on the **View** menu, click **Output** and click the **Debug** tab.
2. Perform one of the following actions to start a debugging session:
   - In the BlackBerry Java Plug-in for Eclipse, right-click the **CustomComponentsDemo** project in the Project Explorer, then on the **Debug As** menu, select **BlackBerry Simulator**.
   - In the BlackBerry JDE, build the **CustomComponentsDemo** project and the **ScreenReaderDemo** project, then on the **Debug** menu, click **Go**.
3. In the BlackBerry Smartphone Simulator, run the AccessibilityDemo application in the **Downloads** folder.

   The simulator displays the Home screen for the application, which is made up of a row of icons, a table, a custom numeric field, and a custom text field.
4. In the BlackBerry Smartphone Simulator, select the numeric field under the table. Press the **U** key on the simulator's keyboard twice to increase the value for the field.

The console window displays the standard output from the assistive technology application (a screen reader). A screen reader listens for events on the UI components and processes the events. The screen reader in the AccessibilityDemo sample application responds to events with `System.out.println()` statements, which the console window displays.

The `println()` statements show the communication between the sample accessible application and the sample screen reader. An actual screen reader responds to the events in an accessible application in a useful way, such as by speaking the text.

```
ScreenReader Context:
com.rim.samples.device.accessibilitydemo.customcomponentsdemo.ValueComponent@fcbe0dd8
---------------- SOUND:  My Gauge Field  value changed to 51
ScreenReader Context:
com.rim.samples.device.accessibilitydemo.customcomponentsdemo.ValueComponent@fcbe0dd8
---------------- SOUND:  My Gauge Field  value changed to 52
```

Related topics

The AccessibilityDemo sample application, 8

Notifying an assistive technology application when the UI changes

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

The notification about the change contains the following information:

- name of the custom UI component
- type of event, for example, the contents of the custom UI component changing or the custom UI component getting focus
- value of the custom UI component before the event
- value of the custom UI component after the event

For example, consider a BlackBerry device application that uses a custom class called `myTextField` that extends the `TextField` class. When a BlackBerry device user changes the text in a `myTextField` instance, an assistive technology application receives a notification and retrieves information about the text that the user changed.

UI changes that trigger a notification to an assistive technology application

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

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

UI component states and properties

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

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

- checked
- editable
- active
- busy

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

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

Provide an assistive technology application with information about a UI change

1. Import the required interface.

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

`AccessibleContext` provides the basic accessibility information about a custom UI component.

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

```
public class MyCustomComponent extends Field
  implements AccessibleContext
{
}
```

3. Create the variables to store the accessibility information for a custom UI component, such as state information.

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

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

```
protected void addAccessibleStates(int states)
{
  _state = _state & ~AccessibleState.UNSET;
  _state = _state | states;
}

protected void removeAccessibleStates(int states)
{
  _state = _state & ~states;
}

public void setAccessibleName(String accessibleName)
```

```
{
  _accessibleName = accessibleName;
}
```

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

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

public AccessibleText getAccessibleText()
{
  return null;
}

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

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

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

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

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

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

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

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

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

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

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

11. Create a method that invokes `AccessibleEventDispatcher.dispatchAccessibleEvent(stateChange, oldState, newState, this)` to send a notification when the state of a custom UI component changes. Pass the following arguments to the method:
    - the event
    - the old state of the custom UI component
    - the new state of the custom UI component
    - an instance of the custom UI component

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

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

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

Provide an assistive technology application with information about text changes

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

1. Import the required interfaces.

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

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

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

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

3. Create the variables to store the information about the custom UI component's state and contents.

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

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

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

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

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

  return null;
}
```

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

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

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

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

7. Implement `AccessibleText.getCaretPosition()` to provide access to the position of the cursor within the text. Return `0` if the user cannot select text.

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

8. Implement `AccessibleText.getSelectionText()` to provide access to the selected part of the text.

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

9. Implement `AccessibleText.getLineCount()` to provide access to the number of lines of text.

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

10. Implement `AccessibleText.getCharCount()` to provide access to the number of characters in the text.

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

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

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

1. Import the required interfaces.

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

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

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

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

3. Create the variables to store the accessibility information about the tabular custom UI component.

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

4. Implement the `get()` methods of `AccessibleTable` to provide the accessibility information about the table.

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

public int getAccessibleRowCount()
{
  return _rowCount;
}

public AccessibleContext[] getAccessibleColumnHeader()
{
  AccessibleContext[] result = new AccessibleContext[_columnNames.length];

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

  return result;
}

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

Provide an assistive technology application with access to numeric values

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

1. Import the required interfaces.

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

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

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

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

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

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

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

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

public int getMaxAccessibleValue()
{
  return _max;
}

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

Enable an assistive technology application to receive notification of UI events

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

1. Import the required interface.

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

2. Create a class that implements `AccessibleEventListener`.

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

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

```
public synchronized void accessibleEventOccurred(
  int event,
  Object oldValue,
  Object newValue,
  AccessibleContext context)
{
  if(context == null)
  {
    return;
  }

  int oldState = (oldValue instanceof Integer) ? ((Integer) oldValue).intValue()
    : 0;
  int newState = (newValue instanceof Integer) ? ((Integer) newValue).intValue()
    : 0;

  switch(context.getAccessibleRole())
  {
    case AccessibleRole.APP_ICON:
      ScreenReaderHandler.handleAppIcon(event, oldState, newState, context);
      break;

    case AccessibleRole.ICON:
      ScreenReaderHandler.handleIcon(event, oldState, newState, context);
      break;

    case AccessibleRole.CHECKBOX:
      ScreenReaderHandler.handleCheckBox(event, oldState, newState, context);
      break;

    case AccessibleRole.CHOICE:
      ScreenReaderHandler.handleChoice(event, oldState, newState, context);
      break;

    ...

  }
}
```

Test an accessible BlackBerry device application

You can create a test screen reader and use it to test and debug your accessible application in BlackBerry® Java® Development Environment 4.6.1 or later or BlackBerry® Java® Plug-in for Eclipse® with BlackBerry Component Package 4.6.1 or later.

Note: Before you make your accessible application available to BlackBerry device users, you should test it on a production screen reader or other assistive technology application that is installed on a BlackBerry device, to verify that your application works the way you want it to.

1. Import the required interface.

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

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

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

3. Implement `accessibleEventOccurred()` to handle the accessible events.

```
public synchronized void accessibleEventOccurred(
  int event,
  Object oldValue,
  Object newValue,
  AccessibleContext context)
{
  switch(context.getAccessibleRole())
  {
    case AccessibleRole.APP_ICON:
      ScreenReaderHandler.handleAppIcon(event, oldState, newState, context);
      break;

    case AccessibleRole.ICON:
      ScreenReaderHandler.handleIcon(event, oldState, newState, context);
      break;

    ...
  }
}
```

4. In the event-handling methods, include `println()` statements to specify the information that you want to generate about the accessible events.

```
static void handleIcon(int event, int oldValue, int newValue, AccessibleContext
context)
{
  if(event == AccessibleContext.ACCESSIBLE_STATE_CHANGED)
  {
    if(Util.hasTransitionedToState( oldValue, newValue, AccessibleState.FOCUSED))
    {
```

```
      String appName = context.getAccessibleName();
      System.out.println("---------------- SOUND: " + appName + " icon focused");
    }
  }
}
```

5. In your accessible application, register the screen reader class as an accessible event listener.

```
_screenReader = new ScreenReader();
AccessibilityManager.setAccessibleEventListener(_screenReader);
```

6. Build the accessible application and the screen reader.
7. Perform one of the following actions to display the console window:
   - In the BlackBerry Java Plug-in for Eclipse, on the **Window** menu, select **Show View > Console**.
   - In the BlackBerry JDE, on the **View** menu, click **Output**, and click the **Debug** tab.
8. Perform one of the following actions to start a debugging session:
   - In the BlackBerry Java Plug-in for Eclipse, right-click the **CustomComponentsDemo** project in the Project Explorer, then on the **Debug As** menu, select **BlackBerry Simulator**.
   - In the BlackBerry JDE, on the **Debug** menu, click **Go**.
9. In the BlackBerry Smartphone Simulator window, test your accessible application.

The console window displays the standard output that comes from the screen reader, which is listening for events on the UI components and processing the events.

```
---------------- SOUND:  Media  icon focused
```

Related topics

The AccessibilityDemo sample application, 8

Related resources

4

For more information about developing accessible applications, see the following resources:

- www.blackberry.com/accessibility
- *BlackBerry Smartphones UI Guidelines*
- Development guides for BlackBerry® Java® application development
- API reference for BlackBerry® Java® application development

Provide feedback

5

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

Document revision history

6

Date	Description
6 April 2010	Updated the following topics: • Introduction to the Accessibility API • The AccessibilityDemo sample application • Set up the AccessibilityDemo sample application in the BlackBerry Java Plug-in for Eclipse • Explore the AccessibilityDemo sample application • Test an accessible BlackBerry device application
6 October 2009	Added the following topics: • Set up the AccessibilityDemo sample application in the BlackBerry JDE Plug-in for Eclipse • Set up the AccessibilityDemo sample application in the BlackBerry Java Development Environment
17 August 2009	Added the following topics: • Best practice: Designing accessible applications • Test an accessible BlackBerry device application
6 July 2009	Initial version.

Legal notice

7

ASSOCIATED WITH ANY APPLICATIONS USED IN CONJUNCTION WITH RIM PRODUCTS OR SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF RIM PRODUCTS OR SERVICES OR ANY PORTION THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR SERVICES, COST OF CAPITAL, OR OTHER SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES WERE FORESEEN OR UNFORESEEN, AND EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

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

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

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with RIM's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for acquiring them. You should not install or use Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and Services that are provided with RIM's products and services are provided as a convenience to you and are provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations, or warranties of any kind by RIM and RIM assumes no liability whatsoever, in relation thereto. Your use of Third Party Products and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other agreements applicable thereto with third parties, except to the extent expressly covered by a license or other agreement with RIM.

Certain features outlined in this documentation require a minimum version of BlackBerry® Enterprise Server, BlackBerry® Desktop Software, and/or BlackBerry® Device Software.

The terms of use of any RIM product or service are set out in a separate license or other agreement with RIM applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY RIM FOR PORTIONS OF ANY RIM PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.

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

Canada

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

Published in Canada