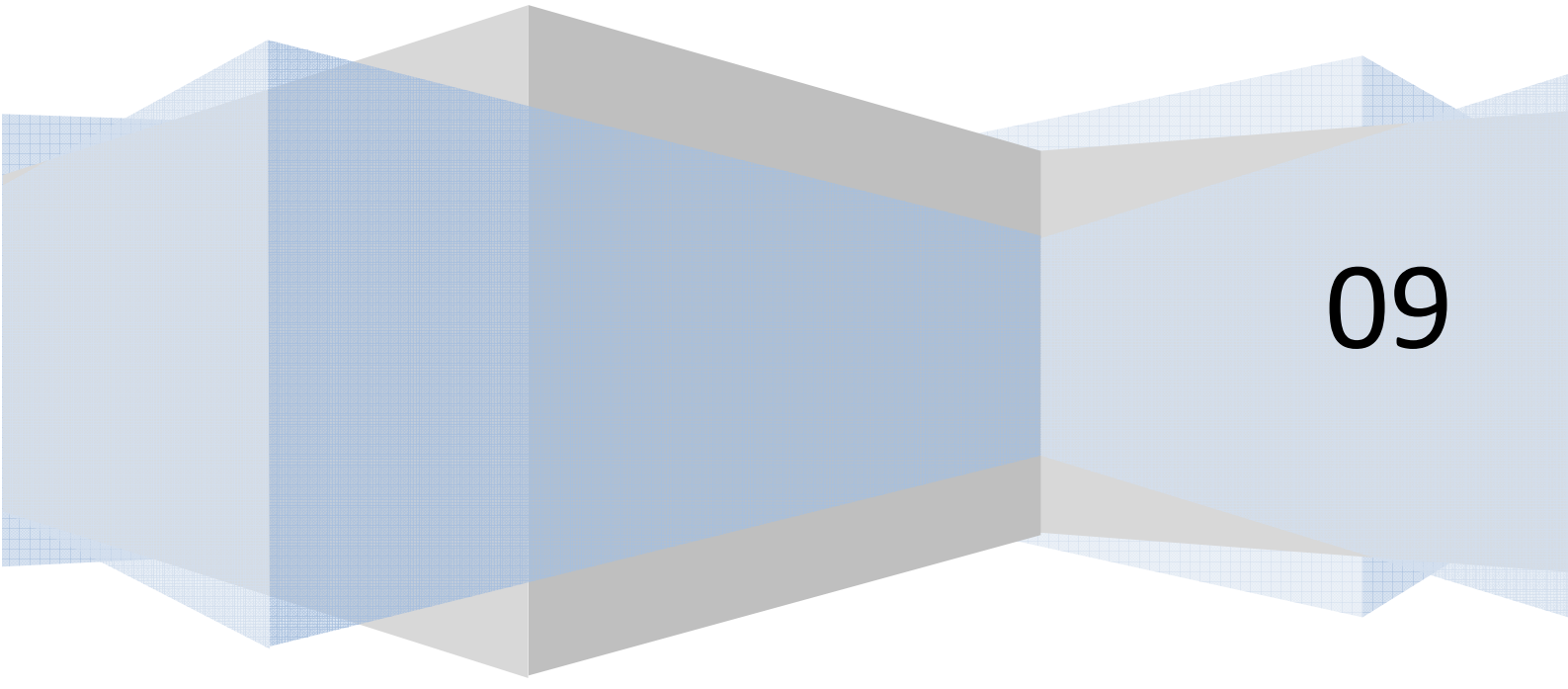
Research In Motion

A11 – User Interface

For BlackBerry SmartPhones

Andre Fabris



09

Contents

A11 User Interface....3
Introduction....4
Setting up New BlackBerry Project....6
Step by step - Interface....7
Title....7
Managers....7
Bitmap Fields....8
Edit and Label Fields....11
Buttons....11
Step by step – MenuItems....13
The whole application....15
Other Variations....18
Links....19

A11 User Interface

This tutorial will show you how create a basic user interface as well as add menu items.

We are going to use Screen, Field, Layout managers and MenuItem classes to create the user interface which is shown on (Figure 1).

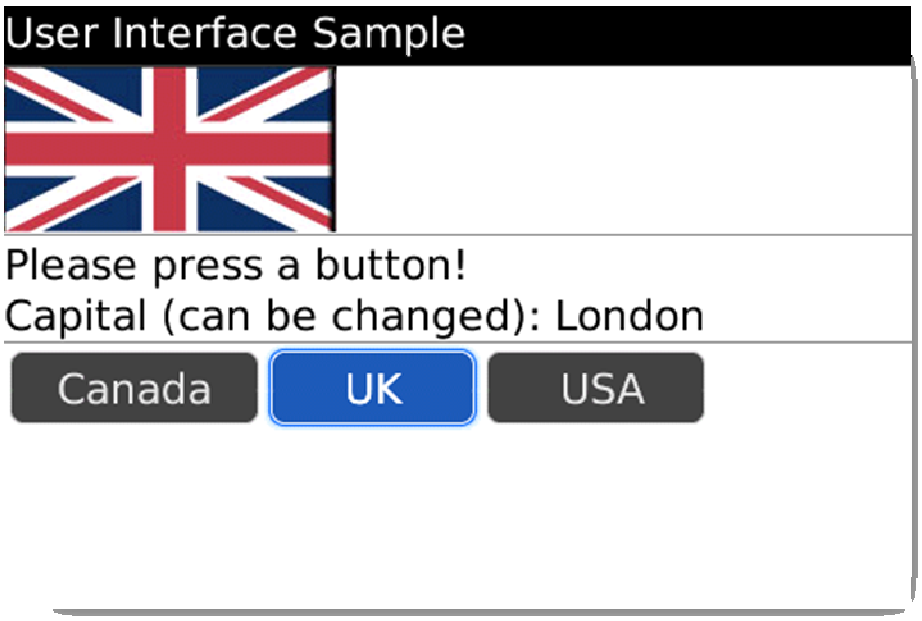


Figure 1

Introduction

You can use standard MIDP APIs and BlackBerry® UI APIs to create BlackBerry® Java Application UIs.

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

- Screen components provide a standard screen layout, a default menu, and standard behaviour when the BlackBerry device user presses the Escape key or clicks the trackwheel or trackball.
- Field components provide standard UI elements for date selection, options, check boxes, lists, text fields and labels, and progress bar controls.
- Layout managers provide an application with the ability to arrange components on a BlackBerry device screen in standard ways, such as horizontally, vertically, or in a left-to-right flow.

Please note that you can always find more information about the APIs we are using in the BlackBerry API reference document which is the part of any JDE Component Pack (Figure 2**Error! Reference source not found.**). You can find it on your computer under Start / Programs / Research in Motion / BlackBerry JDE 4.x.

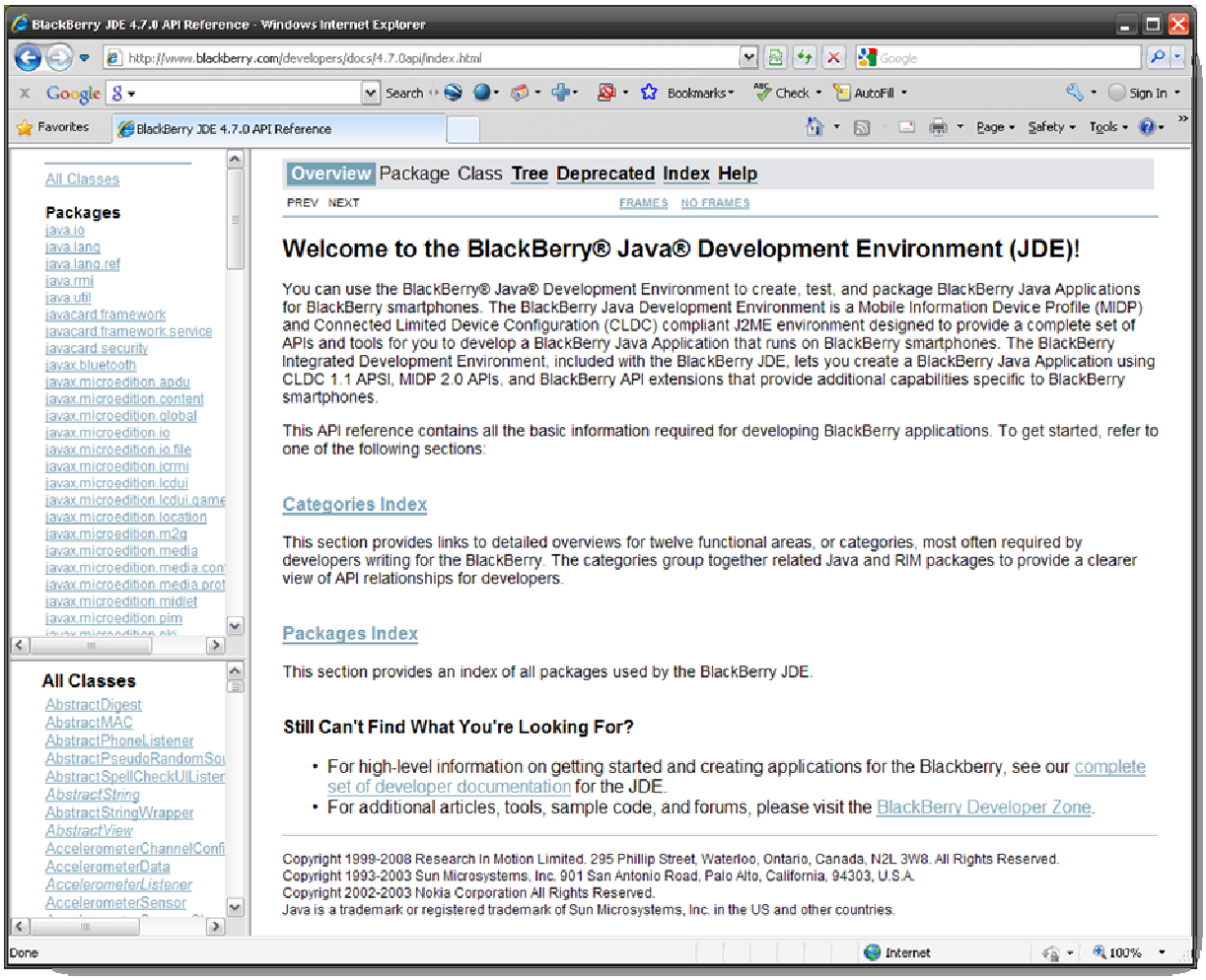

BlackBerry JDE 4.7.0 API Reference - Windows Internet Explorer
http://www.blackberry.com/developers/docs/4.7.0api/index.html
BlackBerry JDE 4.7.0 API Reference
All Classes
Packages
java.io
java.lang
java.lang.ref
java.rmi
java.util
javacard.framework
javacard.framework.service
javacard.security
javax.bluetooth
javax.microedition.apdu
javax.microedition.content
javax.microedition.global
javax.microedition.io
javax.microedition.io.file
javax.microedition.jcrmi
javax.microedition.lcdui
javax.microedition.location
javax.microedition.m2g
javax.microedition.media
javax.microedition.midlet
javax.microedition.pim
All Classes
AbstractDigest
AbstractMAC
AbstractPhoneListener
AbstractSpellCheckUIListen
AbstractString
AbstractStringWrapper
AbstractView
AccelerometerData
AccelerometerListener
AccelerometerSensor
Overview Package Class Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES
Welcome to the BlackBerry® Java® Development Environment (JDE)!
You can use the BlackBerry® Java® Development Environment to create, test, and package BlackBerry Java Applications for BlackBerry smartphones. The BlackBerry Java Development Environment is a Mobile Information Device Profile (MIDP) and Connected Limited Device Configuration (CLDC) compliant J2ME environment designed to provide a complete set of APIs and tools for you to develop a BlackBerry Java Application that runs on BlackBerry smartphones. The BlackBerry Integrated Development Environment, included with the BlackBerry JDE, lets you create a BlackBerry Java Application using CLDC 1.1 APSI, MIDP 2.0 APIs, and BlackBerry API extensions that provide additional capabilities specific to BlackBerry smartphones.
This API reference contains all the basic information required for developing BlackBerry applications. To get started, refer to one of the following sections:
Categories Index
This section provides links to detailed overviews for twelve functional areas, or categories, most often required by developers writing for the BlackBerry. The categories group together related Java and RIM packages to provide a clearer view of API relationships for developers.
Packages Index
This section provides an index of all packages used by the BlackBerry JDE.
Still Can't Find What You're Looking For?
For high-level information on getting started and creating applications for the Blackberry, see our complete set of developer documentation for the JDE.
For additional articles, tools, sample code, and forums, please visit the BlackBerry Developer Zone.
Copyright 1999-2008 Research In Motion Limited. 295 Phillip Street, Waterloo, Ontario, Canada, N2L 3W8. All Rights Reserved.
Copyright 1993-2003 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
Copyright 2002-2003 Nokia Corporation All Rights Reserved.
Java is a trademark or registered trademark of Sun Microsystems, Inc. in the US and other countries.
Done
Internet
100%

Figure 2

Setting up New BlackBerry Project

We have explained how to setup and configure a new BlackBerry project in A10 tutorial.

Next, we will create a new project called User Interface and a new class which extends UiApplication UserInterface.

In the same file we can add UserInterFaceScreen and extend MainScreen.

Every application that shows UI interface must have at least one Screen object.

```
public class UserInterface extends UiApplication {
	public static void main(String[] args) {
		UserInterface theApp = new UserInterface();
		theApp.enterEventDispatcher();
	}
	public UserInterface() {
		pushScreen(new UserInterfaceScreen());
	}
}

final class UserInterfaceScreen extends MainScreen {
	HorizontalFieldManager _fieldManagerTop;
	VerticalFieldManager _fieldManagerMiddle;
	HorizontalFieldManager _fieldManagerBottom;
	BitmapField _bitmap;
	Bitmap _canadaImage, _ukImage, _usImage;
	LabelField _label;
	BasicEditField _input;
	String _canadaCapital, _ukCapital, _usCapital, _capital;
	int displayed = 0;

	public UserInterfaceScreen() {
		super();
```

The UserInterface class is a standard entry point into our application explained in the A10 tutorial. UserInterfaceScreen extends MainScreen and the same application layout was again used in A10. The variable declaration follows, and what each one of these members do will be explained shortly.

Step by step - Interface

Title

We have seen in the A10 tutorial how to setup a title. MainScreen has the setTitle(Field title); method which performs this function.

```
LabelField title = new LabelField("User Interface
Sample",LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
setTitle(title);
```

Managers

We would like to divide our screen into three zones: top, middle and bottom. You can divide the screen in as many parts as you want in your application, or not. We use vertical and horizontal managers.

These managers will align all the added components into a single column or row depending on their orientation.

For this example, we will use a top and bottom as a horizontal manager and a middle one as a vertical manager.

Once we create managers we need to add them in the order we want to display them on the screen. The screen itself has a vertical manager built in.

```
HorizontalFieldManager _fieldManagerTop;
VerticalFieldManager _fieldManagerMiddle;
HorizontalFieldManager _fieldManagerBottom;

fieldManagerTop = new HorizontalFieldManager();
_fieldManagerMiddle = new VerticalFieldManager();
_fieldManagerBottom = new HorizontalFieldManager();
add(_fieldManagerTop);
add(new SeparatorField());
add(_fieldManagerMiddle);
add(new SeparatorField());
add(_fieldManagerBottom);
```

Note that we used `SeparatorField` to draw a line between our sections. That is not required, however it does make the screen look nicer.

Bitmap Fields

To display bitmaps (we use three flags: Figure 3) we use the `BitmapField` and `Bitmap` object.

Loading images and creating a new bitmap field is quite simple. Once it is created we set the image we want to display in the field. Our default value is the Canadian Flag.

We want to add the bitmap field to the top manager and we use it with an add function.

```
BitmapField _bitmap;
Bitmap _canadaImage, _ukImage, _usImage;

_canadaImage = Bitmap.getBitmapResource("canada.png");
_ukImage = Bitmap.getBitmapResource("uk.png");
_usImage = Bitmap.getBitmapResource("us.png");
_bitmap = new BitmapField();
_bitmap.setBitmap(_canadaImage);
_fieldManagerTop.add(_bitmap);
```

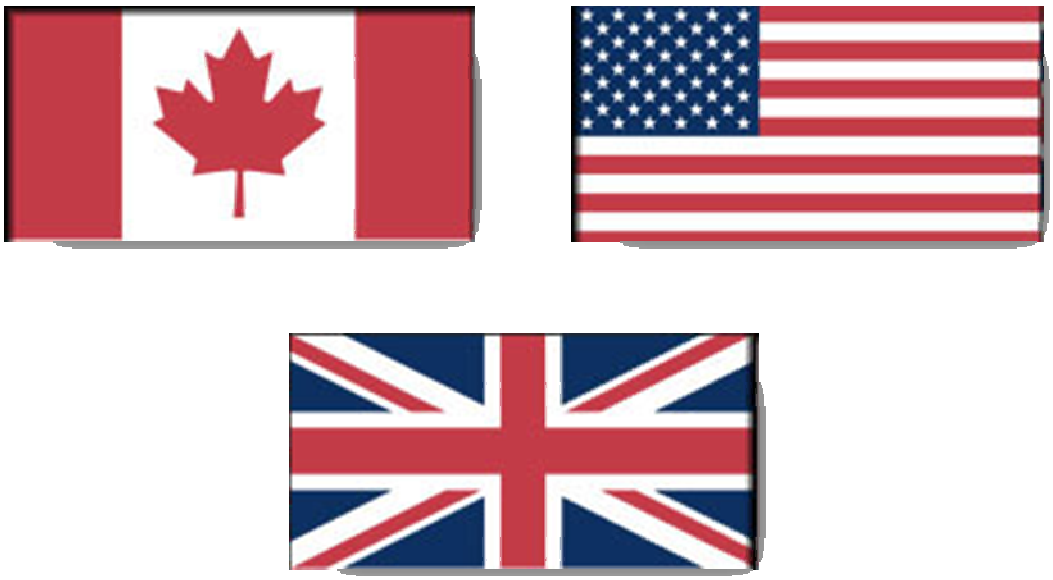


Figure 3

To be able to use images in our project we need to import them.

Clicking on File/Import will bring you the import menu (Figure 4).

As you can see we can import from many different srouces into our project but we want to select File System

Press next, browse for directory and select required files (Figure 5).

In our case those were canada.png, us.png and uk.png. Be careful java is CaSe SeNsItIvE.

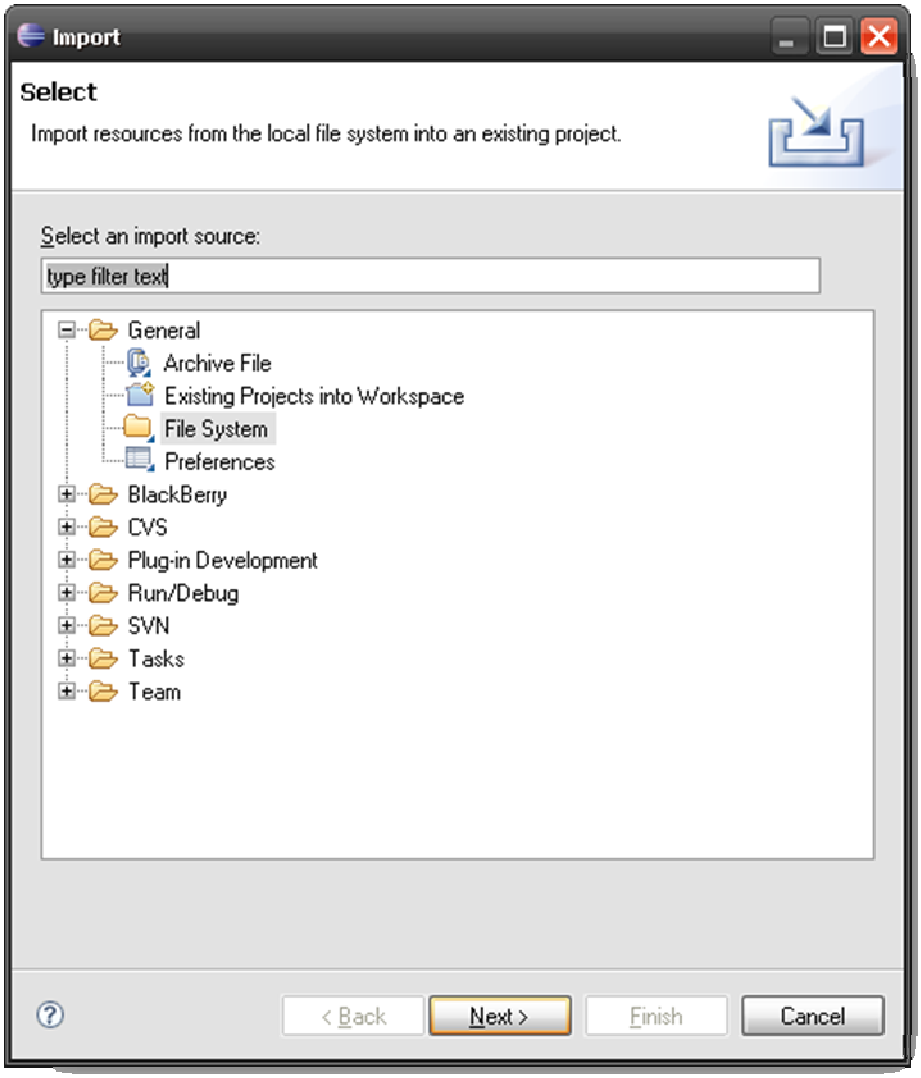


Figure 4

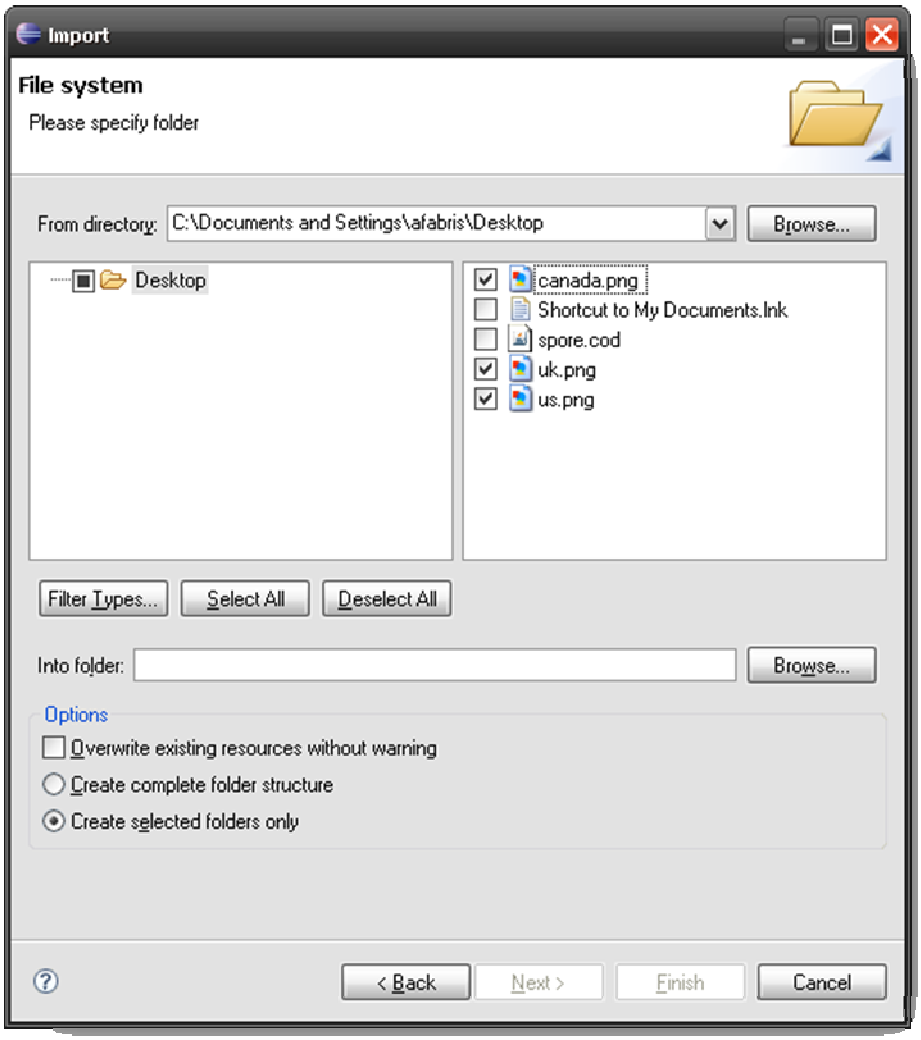

Import
File system
Please specify folder
From directory: C:\Documents and Settings\afabris\Desktop
Browse...
Desktop
canada.png
Shortcut to My Documents.lnk
spore.cod
uk.png
us.png
Filter Types...
Select All
Deselect All
Into folder:
Browse...
Options
Overwrite existing resources without warning
Create complete folder structure
Create selected folders only
< Back
Next >
Finish
Cancel

Figure 5

Edit and Label Fields

The Middle manager will have one label which the user cannot change and one which can be edited:

```
LabelField _label;
BasicEditField _input;
String _canadaCapital, _ukCapital, _usCapital, _capital;
_canadaCapital = "Ottawa";
_ukCapital = "London";
_usCapital = "Washington";
_capital = _canadaCapital;
_label = new LabelField("Please press a button!");
_input = new BasicEditField("Capital (can be changed): ", _capital);
_fieldManagerMiddle.add(_label);
_fieldManagerMiddle.add(_input);
```

Buttons

We will add three buttons in the bottom part of our application. Those buttons will be called "Canada", "UK" and "USA" and will be used to change the information on the screen – displaying the flag and the capital.

To create buttons all you need to do is a simple call:

```
ButtonField canadaButton = new ButtonField("Canada");
ButtonField ukButton = new ButtonField("  UK  ");
ButtonField usButton = new ButtonField("  USA ");
```

Some spaces are added around the UK and USA words just to make the buttons a bit wider.

These buttons at the moment have no functionality. What we would like to do is to have a listener to detect when the buttons are pressed and then take action. We use `FieldChangeListener` which can listen to any changes on any field. We will need to implement the `fieldChanged` method.

```
FieldChangeListener listenerCanada = new FieldChangeListener() {
      public void fieldChanged(Field field, int context) {
      _bitmap.setBitmap(_canadaImage);
      _input.setText(_canadaCapital);
      displayed = 0;
      }
};

FieldChangeListener listenerUK = new FieldChangeListener() {
      public void fieldChanged(Field field, int context) {
      _bitmap.setBitmap(_ukImage);
      _input.setText(_ukCapital);
```

```
        displayed = 1;
        }
};

FieldChangeListener listenerUS = new FieldChangeListener() {
        public void fieldChanged(Field field, int context) {
        _bitmap.setBitmap(_usImage);
        _input.setText(_usCapital);
        displayed = 2;
        }
};
```

For simplicity we have three listeners, which can be avoided. Each listener overrides the fieldChanged method and sets the image we want to display, as well as the name of the capital.

I use a variable called “displayed” of type int to track which country is currently displayed. We will use that information a little bit later.

The last thing we need to do is assign listeners to the `ButtonFields`:

```
canadaButton.setChangeListener(listenerCanada);
ukButton.setChangeListener(listenerUK);
usButton.setChangeListener(listenerUS);
```

And at the end, add the buttons to the `Manager`.

```
_fieldManagerBottom.add(canadaButton);
_fieldManagerBottom.add(ukButton);
_fieldManagerBottom.add(usButton);
```

Step by step – MenuItems

Now that we have the interface created, we can add some menu items to our menu. We will add two: “Close” which we will use to exit the application and “Change Capital”. The user can edit the name of the capital city and then click on the “Change Capital” menu item. That action will save the name entered and display the new capital name from that moment on.

To add items to the menu (which is displayed by pressing the device menu key) (Figure 6), we will need to overwrite the makeMenu method of our screen:

```
protected void makeMenu(Menu menu, int instance) {
      menu.add(_changeCapital);
      menu.add(_close);
}
```

We also need to define the menu items (`_changeCapital` and `_close`) and setup their functionality.

Each menu item needs to have its run method implemented to be able to run those actions once the user selects that item.

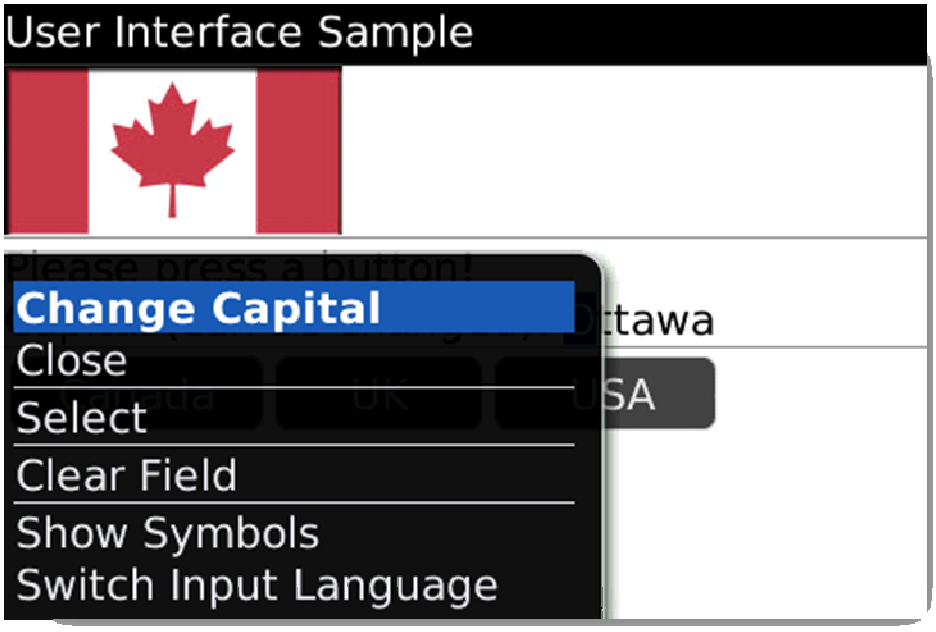


Figure 6

```
private MenuItem _close = new MenuItem("Close", 110, 10) {
      public void run() {
            onClose();
      }
};
```

The “Close” button will call the `onClose()` method –we used this method in the A10 tutorial – recall, it displays a goodbye dialog.

```
public boolean onClose() {
      Dialog.alert("Goodbye!");
      System.exit(0);
      return true;
}
```

The two numbers (110, 10) which we used in creating the menu item represent order and priority. If the order numbers are less then 0x00010000apart they will be grouped together. A lower value passed into this method for priority indicates a higher priority. The priority value applies to the default menu item.

Here is the code for the other menu item:

```
private MenuItem _changeCapital = new MenuItem("Change Capital", 110, 10) {
      public void run() {
      if (displayed == 0)
            _canadaCapital = _input.getText();
      else if (displayed == 1)
            _ukCapital = _input.getText();
      else if (displayed == 2)
            _usCapital = _input.getText();
      }
};
```

We are using the variable “displayed” to determine which capital the user is editing and then we replace the name of that capital with whatever value is in “_input field”. The next time the user selects that country the new value will be displayed.

The whole application

```
package com.rim.samples.userinterface;

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

public class UserInterface extends UiApplication {

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

	public UserInterface() {
		pushScreen(new UserInterfaceScreen());
	}
}

final class UserInterfaceScreen extends MainScreen {
	HorizontalFieldManager _fieldManagerTop;
	VerticalFieldManager _fieldManagerMiddle;
	HorizontalFieldManager _fieldManagerBottom;
	BitmapField _bitmap;
	Bitmap _canadaImage, _ukImage, _usImage;
	LabelField _label;
	BasicEditField _input;
	String _canadaCapital, _ukCapital, _usCapital, _capital;
	int displayed = 0;

	public UserInterfaceScreen() {
		super();

	LabelField title = new LabelField("User Interface Sample",
LabelField.ELLIPSIS | LabelField.USE_ALL_WIDTH);
setTitle(title);

	_fieldManagerTop = new HorizontalFieldManager();
	_fieldManagerMiddle = new VerticalFieldManager();
	_fieldManagerBottom = new HorizontalFieldManager();
		add(_fieldManagerTop);
		add(new SeparatorField());
		add(_fieldManagerMiddle);
		add(new SeparatorField());
		add(_fieldManagerBottom);

	_canadaImage = Bitmap.getBitmapResource("canada.png");
	_ukImage = Bitmap.getBitmapResource("uk.png");
	_usImage = Bitmap.getBitmapResource("us.png");
	_bitmap = new BitmapField();
	_bitmap.setBitmap(_canadaImage);
	_fieldManagerTop.add(_bitmap);

	_canadaCapital = "Ottawa";
	_ukCapital = "London";
	_usCapital = "Washington";
	_capital = _canadaCapital;
```

```
        _label = new LabelField("Please press a button!");
        _input = new BasicEditField("Capital (can be changed): ", _capital);
        _fieldManagerMiddle.add(_label);
        _fieldManagerMiddle.add(_input);

        FieldChangeListener listenerCanada = new FieldChangeListener() {
        public void fieldChanged(Field field, int context) {
        _bitmap.setBitmap(_canadaImage);
        _input.setText(_canadaCapital);
        displayed = 0;
        }
};
        FieldChangeListener listenerUK = new FieldChangeListener() {
        public void fieldChanged(Field field, int context) {
        _bitmap.setBitmap(_ukImage);
        _input.setText(_ukCapital);
        displayed = 1;
}
};
        FieldChangeListener listenerUS = new FieldChangeListener() {
        public void fieldChanged(Field field, int context) {
        _bitmap.setBitmap(_usImage);
        _input.setText(_usCapital);
        displayed = 2;
}
};

        ButtonField canadaButton = new ButtonField("Canada");
        ButtonField ukButton = new ButtonField("  UK  ");
        ButtonField usButton = new ButtonField("  USA ");
              canadaButton.setChangeListener(listenerCanada);
              ukButton.setChangeListener(listenerUK);
              usButton.setChangeListener(listenerUS);
              _fieldManagerBottom.add(canadaButton);
              _fieldManagerBottom.add(ukButton);
              _fieldManagerBottom.add(usButton);
        }

        protected void makeMenu(Menu menu, int instance) {
              menu.add(_changeCapital);
              menu.add(_close);
        }

        private MenuItem _changeCapital = new MenuItem("Change Capital", 110,
10) {
        public void run() {
        if (displayed == 0)
              _canadaCapital = _input.getText();
        else if (displayed == 1)
              _ukCapital = _input.getText();
        else if (displayed == 2)
              _usCapital = _input.getText();
        }
        };

        private MenuItem _close = new MenuItem("Close", 110, 10) {
        public void run() {
              onClose();
        }
        };
```

```
        public boolean onClose() {
                Dialog.alert("Goodbye!");
                System.exit(0);
                return true;
        }
}
```

Other Variations

There are number of things you might want to try with this sample application:

- Try to change titles and messages
- Try to add other fields from `net.rim.device.api.ui.component` package such as SeparatorField
- Try to have only one listener for the buttons.

It would also be nice if we didn't have to use the menu item "Change Capital" and have the application listen for changes in "`_input`" field and update the data accordingly.

Links

BlackBerry Developers Web Site:

http://na.blackberry.com/eng/developers/

Developer Video Library:

- Introduction to BlackBerry Development:

  http://www.blackberry.com/DevMediaLibrary/view.do?name=IntroBlackBerryDev

- Introduction to BlackBerry Java Development:

  http://www.blackberry.com/DevMediaLibrary/view.do?name=java

Developer Labs:

- Introduction to User Interface managers
- Displaying a list of complex records

  http://na.blackberry.com/eng/developers/resources/developer_labs.jsp#tab_tab_jde

Documentation:

- Documentation for developers can be found here:

  http://na.blackberry.com/eng/support/docs/developers/?userType=21

Knowledge Base Articles:

- Number of additional useful documents can be found here:

  http://www.blackberry.com/knowledgecenterpublic/livelink.exe/fetch/2000/348583/customview.html?func=ll&objId=348583

Forums:

- The link to BlackBerry Development Forums:

  http://supportforums.blackberry.com/rim/?category.id=BlackBerryDevelopment