

Research In Motion

A12 – Localizing

For BlackBerry SmartPhones

Andre Fabris



09

Contents

A12 Localizing.....	4
Introduction.....	6
Setting up a New BlackBerry Project.....	7
Creating resource files.....	9
StoreInfo class	17
The whole application	19
Other Variations	20
Links.....	21



A12 Localizing

This tutorial will show you how to localize your application (Figure 1). We will use our Hello World application and make it international – we will translate it in French (Figure 2) and Spanish (Figure 3).

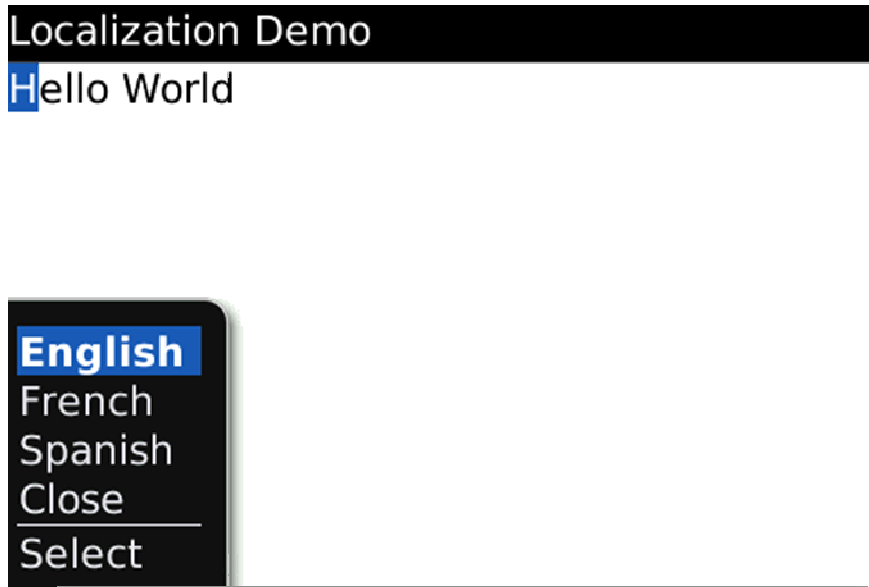


Figure 1



Figure 2

Note that we translated the menu as well as title. We even translated the dialogues.

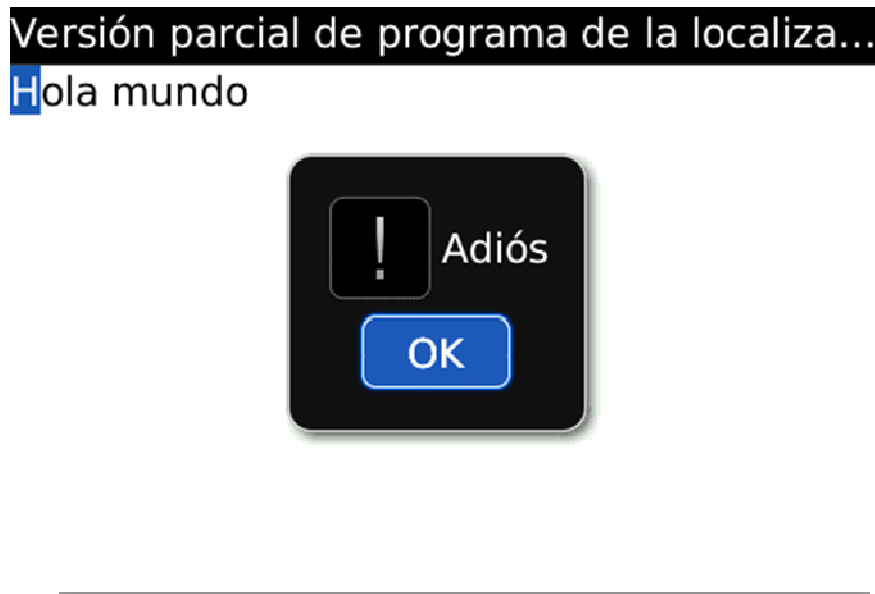


Figure 3

Introduction

Any serious commercial application these days should be localized. BlackBerry uses `net.rim.device.api.i18n` package to enable seamless localization.

If your application is localized it will be displayed in the local language out of the box. So if the default locale settings on one device are set to English and on the other French, the application will be displayed in the correct language on both devices without any need for any interaction from the user.

Our example shows, for demonstration purposes, how to change the Locale setting manually.

Locale settings allows us not just to display the interface of our application in the correct language, but also formats dates, numbers, etc. For more information check out the BlackBerry API reference document which is the part of any JDE Component Pack (Figure 4). You can find it on your computer under Start / Programs / Research in Motion / BlackBerry JDE 4.x.

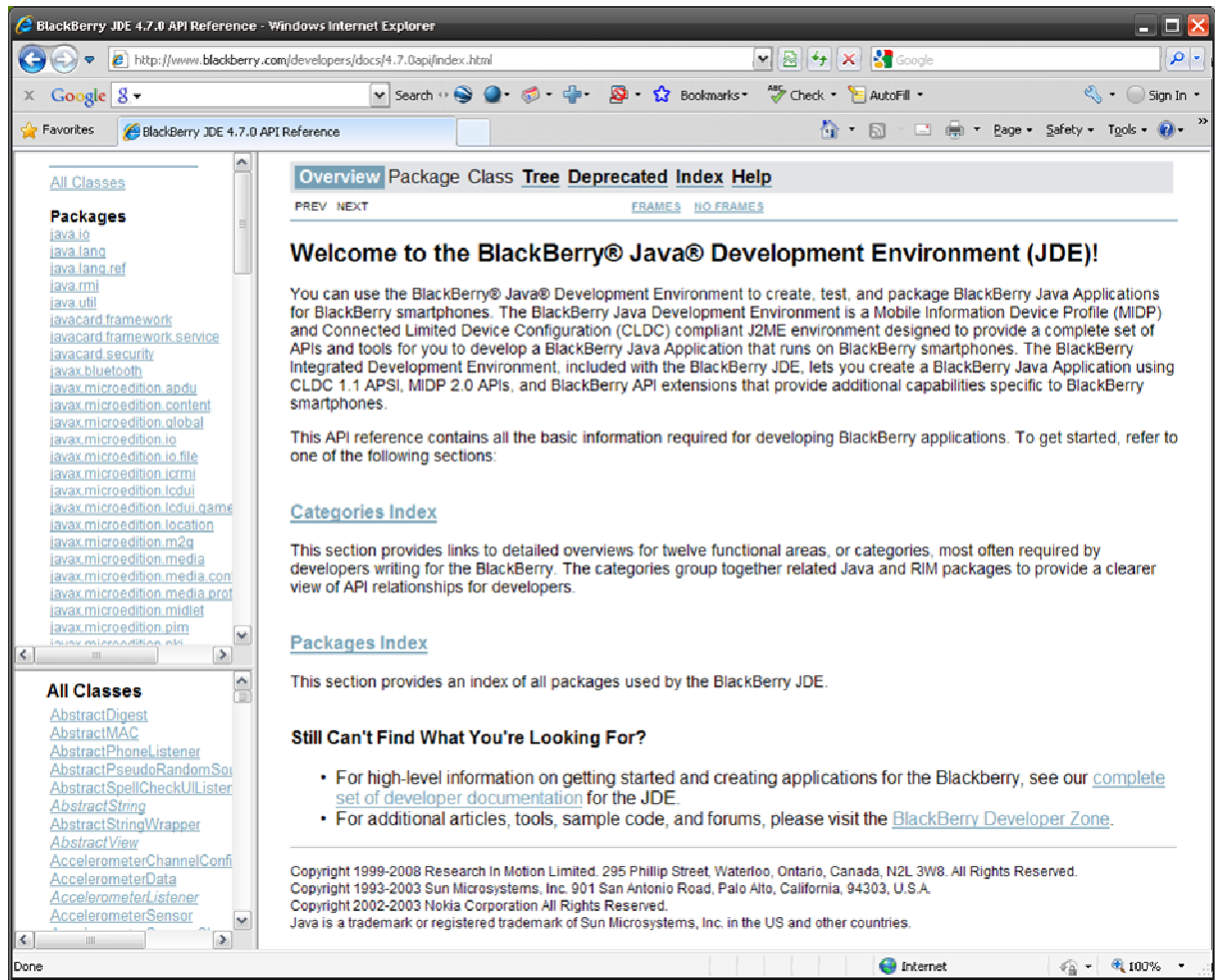


Figure 4

Setting up a New BlackBerry Project

You should already know how to setup and configure a new BlackBerry project. If you need a reminder, please refer to the A10 tutorial. I will also assume you know how to create menu and display items. If not please refer to the A11 tutorial.

Here is the start of our application:

```
package com.rim.samples.local;

import net.rim.device.api.ui.MenuItem;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.i18n.*;

public class Local extends UiApplication {
    public static void main(String[] args) {
        Local theApp = new Local();
    }
}
```



```
        theApp.enterEventDispatcher();
    }

    public Local() {
        pushScreen(new LocalScreen());
    }
}

final class LocalScreen extends MainScreen implements LocalDemoResource {

    private static ResourceBundle _res =
ResourceBundle.getBundle(BUNDLE_ID, BUNDLE_NAME);
    LabelField title;
    RichTextField rtf;

    public LocalScreen() {
        super();
        title = new LabelField(_res.getString(FIELD_TITLE),
LabelField.ELLIPSIS| LabelField.USE_ALL_WIDTH);
        setTitle(title);
        rtf = new RichTextField(_res.getString(MESSAGE));
        add(rtf);
    }
}
```

If you read the code carefully you will notice that our `LocalScreen` class implements `LocalDemoResource`. Also new is the `ResourceBundle` variable `_res` which is used when creating our fields.

To be able to use and explain them we will first create our resource files.

Creating resource files

To be able to localize your application you need to create .rrc and .rrh files.

It is quite easy to create these files in Eclipse: Click on File / New and then select Other.

You need to select BlackBerry Resource File (Figure 5), and press Next.

On the next screen you need to browse to the location of your package. In our case it is Local/src/com/rim/local (Figure 6).

Let's call this file LocalDemo.rrh (make sure you include the extension).

Press Finish. This will now allow you to implement LocalDemoResource interface.

If you called your .rrh file example exampleResource.rrh the interface name would be exampleResource.

Now we need to create resource files for each language: for default language: LocalDemo.rrc, French: LocalDemo_fr.rrc and Spanish: LocalDemo_es.rrc.

The parts of the name _fr and _es are default endings for those languages and cannot be changed.

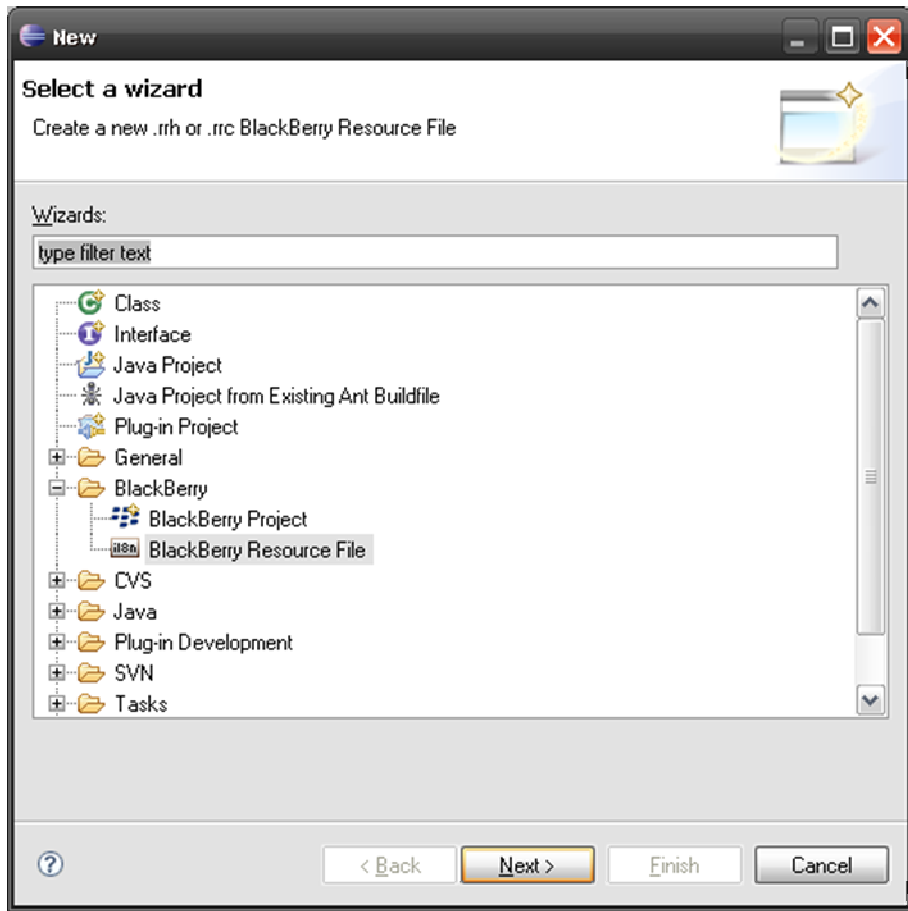


Figure 5

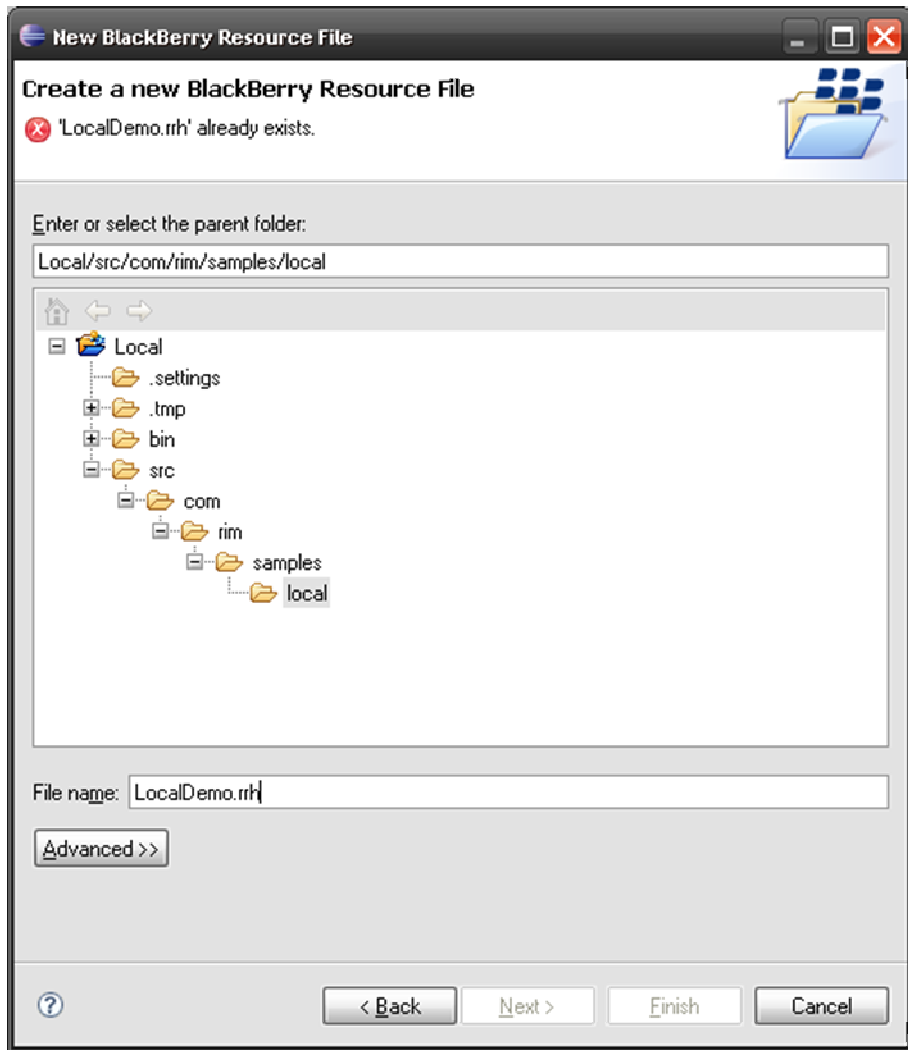


Figure 6

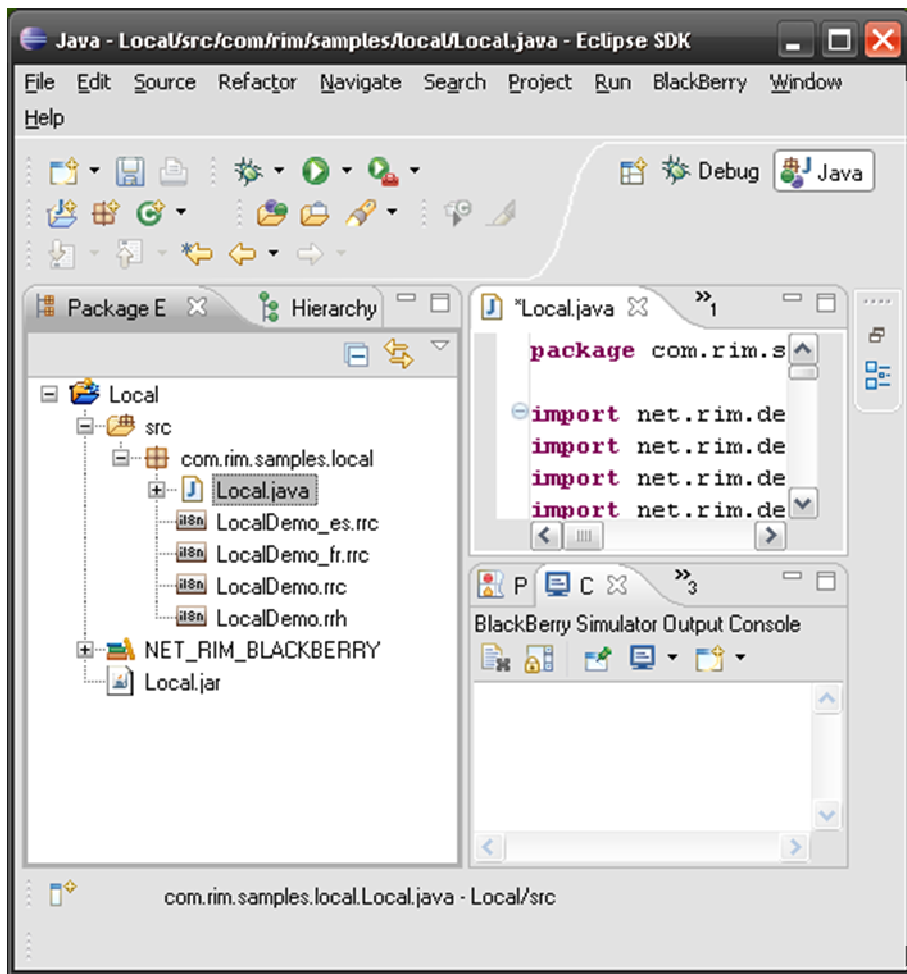



Figure 7

If you did everything correctly you should have 4 resource files just under your Local.java file. Note the  icon in front of the file names (Figure 7).

Now we need to edit these files. Double click on the LocalDemo.rh file. This is the one which contains all the keys:

Select the 'Add Key' Button and add the following fields

CLOSE, ENGLISH, FIELD_TITLE, FRENCH, GOODBYE, MESSAGE, SPANISH.

You do not need to enter the values on this page (Figure 8).

When you double click on the LocalDemo.rcc file it will open the screen as shown on (Figure 9). Fill in the values as shown on the picture. You can access the values for French and Spanish locales by clicking on the tab on the bottom of the window (marked fr or es) or by double clicking on the file name. Fill those out as shown on (Figure 10 and Figure 11).

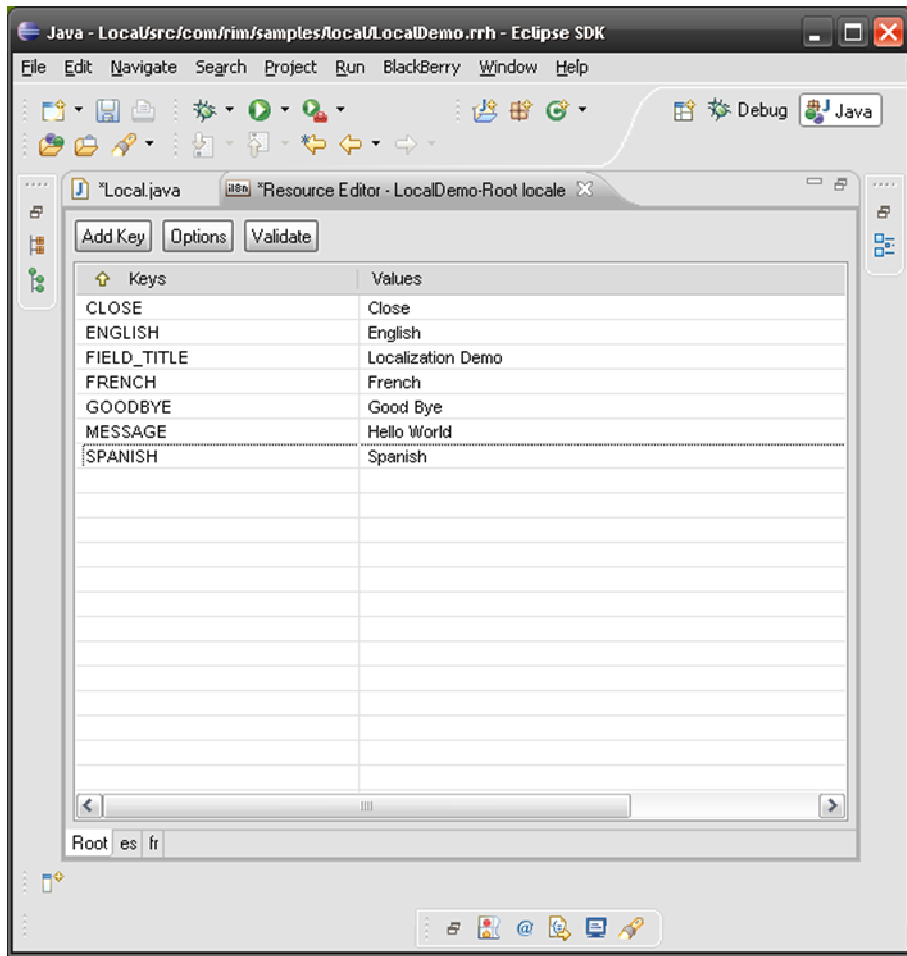


Figure 9

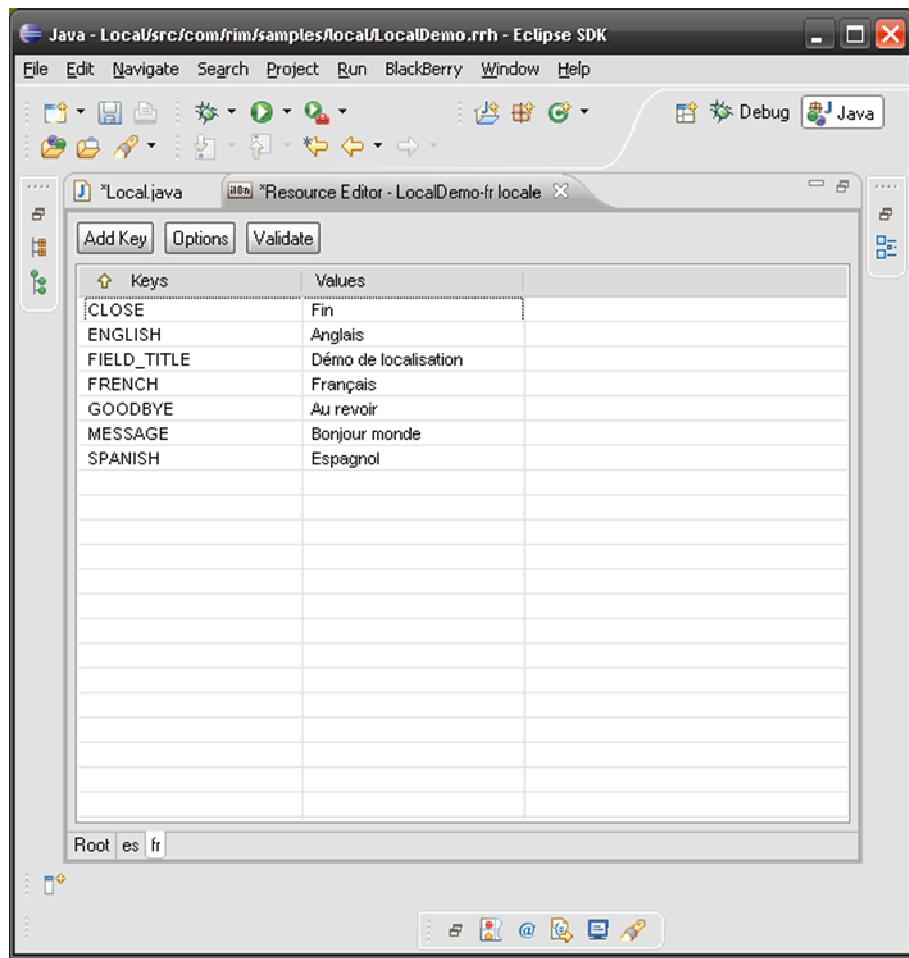


Figure 10

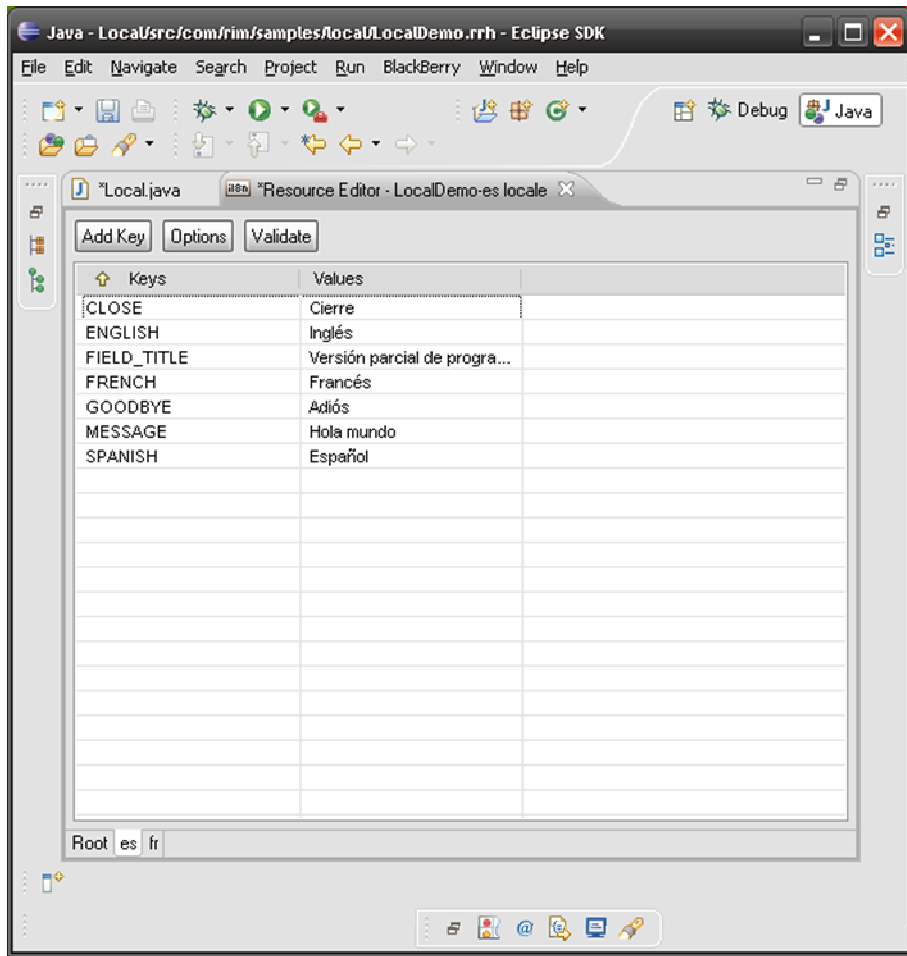


Figure 11

As you can see, special characters used in these languages such as 'ó' in word Adiós or 'ñ' in Español were used. The system will use those characters and display them when needed.

You can use any Unicode characters here, so this means you can support languages such as Chinese or Arabic.

As you can see the field names such as FIELD_TITLE or MESSAGE were used when creating our fields:

```
RichTextField(_res.getString(MESSAGE));
add(rtf);
```


StoreInfo class

The rest of our application is quite simple

```
public boolean onClose() {
    Dialog.alert(_res.getString(GOODBYE));
    System.exit(0);
    return true;
}
```

Note that Dialog.alert() now uses a localized version.

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

private MenuItem _close = new MenuItem(_res.getString(CLOSE), 110, 10) {
    public void run() {
        onClose();
    }
};

private MenuItem _english = new MenuItem(_res.getString(ENGLISH), 110, 10) {
    public void run() {
        Locale.setDefault (Locale.get(Locale.LOCALE_en, null));
        refresh();
    }
};

private MenuItem _french = new MenuItem(_res.getString(FRENCH), 110, 10) {
    public void run() {
        Locale.setDefault (Locale.get(Locale.LOCALE_fr, null));
        refresh();
    }
};

private MenuItem _spanish = new MenuItem(_res.getString(SPANISH), 110, 10) {
    public void run() {
        Locale.setDefault (Locale.get(Locale.LOCALE_es, null));
        refresh();
    }
};
```

Creating menu items is also simple. Note that we used the Locale.setDefault() call to change the current Locale settings the application is using.

This is done for demonstration purposes only; you will not need to set these values manually in your application.



At the end we call the refresh() function to update the values on our screen:

```
private void refresh() {
    title.setText(_res.getString(FIELD_TITLE));
    deleteAll();
    rtf = new RichTextField(_res.getString(MESSAGE));
    add(rtf);
    _english.setText(_res.getString(ENGLISH));
    _french.setText(_res.getString(FRENCH));
    _spanish.setText(_res.getString(SPANISH));
    _close.setText(_res.getString(CLOSE));
}
```

Remember, you do not need to refresh your application in your application; this is done only because the example changes the Locale language while it is running.

The whole application

It is not convenient to attach the listing of the whole application, but all its parts are displayed in the text above.

However, here is the content of the LocalDemo.rrh file:

```
package com.rim.samples.local;

FIELD_TITLE#0=0;
MESSAGE#0=1;
GOODBYE#0=2;
CLOSE#0=3;
ENGLISH#0=4;
SPANISH#0=5;
FRENCH#0=6;
```

And the Spanish .rrc file:

```
CLOSE#0="Cierre";
ENGLISH#0="Inglés";
FIELD_TITLE#0="Versión parcial de programa de la localización";
FRENCH#0="Francés";
GOODBYE#0="Adiós";
MESSAGE#0="Hola mundo";
SPANISH#0="Español";
```

As you can see there are no secrets here and you can edit these files in any text editor once you create them in Eclipse.

Other Variations

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

- Try to add more languages.
- Try to add different field types such as

```
FIELD COUNTRIES#0={  
    "United States",  
    "China",  
    "Germany",  
};
```

- Use numbers, dates and currencies to see how these are displayed.

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:

- Localizing an application

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:

- What Is - BlackBerry support for multiple languages and localization in an application
- How To - Set the BlackBerry Simulator Keypad Locale field

<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>

Sample Code:

- There is a sample application located in your JDE folder i.e.:
C:\Program Files\Research In Motion\ BlackBerry JDE 4.6.0
\samples\com\rim\samples\device\localizationdemo

