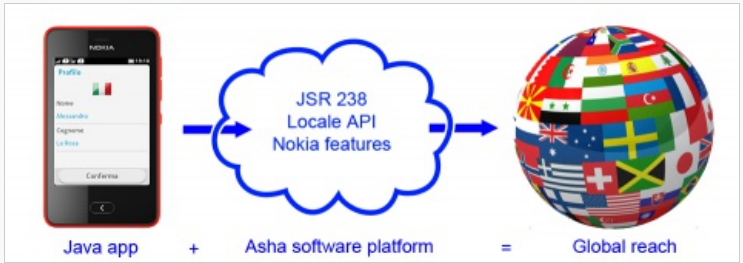
Localizing a Java app on the Asha software platform

This article explains how to localize a Java app on the Asha software platform using features offered by the Mobile Internationalization API, the Locale API and Nokia-specific features like MIDlet name localization.

23 Jun 2013

Introduction

The Asha software platform introduces support for the Mobile Internationalization API (JSR 238) that, together with other features supported by the platform, allows to fully localize a Java app.



Localization features on the Asha software platform

This article illustrates the available features, showing how each of them can be used to add localization support, so increasing the potential reach of a Java app.

Localizing app resources

Localization of an app typically involves both textual and binary data (for instance, image, video and audio files). This process requires two steps:

- creating the localized resources for all the supported locales
- letting the app use the proper localized resources depending on the default device locale

The resource localization process is made simple by using the ResourceMaker tool provided by Nokia. This tool allows you to define localized resources in an XML file that are then converted to the necessary file formats needed by the *Internationalization API* to properly handle localization within the Java app.

The resource XML file

Before starting, it is necessary to create the XML resource file.

Note: The resource file is needed by the *ResourceMaker* tool in order to generate the *Internationalization API* needed files.

For this reason, since it is not directly used by the Java app itself, the XML file **should not be placed within the project source folder**. A possible location could be the root of a Java project, or a sub-folder not included in the project build path.

The following code snippet shows the skeleton of an empty XML resource file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources SYSTEM "resources.dtd">
<resources version="1.0">

</resources>
```

Resource sets

An XML resource file can contain one or more resource sets. Each resource set contains a logically grouped set of resources, that will be accessed by the Java app using a common **base name**. Each base name will be converted to a different Java class by the *ResourceMaker* tool, as explained in the dedicated section below. For this reason, each resource sets *must have a different base name*, and the base name must follow the naming conventions for Java classes.

The following code snippet shows an XML resource file containing two resource sets.

```
<?xml version="1.0"?>
```


(C) Copyright Nokia 2014. All rights reserved.

```
<!DOCTYPE resources SYSTEM "resources.dtd">
<resources version="1.0">  <!-- JSR-238 version number -->
    <set basename="MyAppLocale">

    </set>

    <set basename="MySecondAppLocale">

    </set>
</resources>
```

The `set` element will contain all the localized versions of the set resources. The following sections show how to provide localized versions of both textual and binary data.

Localizing text resources

Textual resources are defined by a `<resource>` XML element with the following attributes:

- `type`: must be valued `string`
- `id`: must have an integer value, unique within the resource set
- `name`: must have a string value, unique within the resource set, and following the conventions for Java constant values

Each `<resource>` element must contain a `<value>` child element for each localization of the textual resource. Each `<value>` element must:

- specify the locale within a `locale` attribute
- contain the localized resource value as textual child of the element

The following code snippet shows two text resources localized in English and Italian (Italy) language:

```
<?xml version="1.0"?>
<!DOCTYPE resources SYSTEM "resources.dtd">
<resources version="1.0">  <!-- JSR-238 version number -->
    <set basename="MyAppLocale">
        <resource id="1" type="string" name="FIRST_NAME">
            <value locale="en">First name</value>
            <value locale="it-IT">Nome</value>
        </resource>

        <resource id="2" type="string" name="LAST_NAME">
            <value locale="en">Last name</value>
            <value locale="it-IT">Cognome</value>
        </resource>
    </set>
</resources>
```

Localizing binary resources

As seen for textual resources, binary resources are similarly defined by a `<resource>` element, specifying both a unique `id` and `name` attribute, and with a `type` attribute valued `binary`. The `<resource>` element must then contain a `<value>` child element for each localized version of the resource, with each of those specifying:

- a `locale` attribute specifying the locale for the specific resource version
- a `file` attribute specifying the path to the binary file

The path must be specified as relative to a common base directory. This base directory will be then specified when executing the *ResourceMaker* tool, so that binary resources can be properly accessed.

(C) Copyright Nokia 2014. All rights reserved.

The following code snippet shows the definition of a localized binary resource:

```
<?xml version="1.0"?>
<!DOCTYPE resources SYSTEM "resources.dtd">
<resources version="1.0">
    <set basename="MyAppLocale">

        <resource id="8" type="binary" name="FORM_ICON">
   <value locale="en" file="assets/t_usa.png"/>
   <value locale="it-IT" file="assets/t_italy.png"/>
        </resource>

    </set>
</resources>
```

Note: The binary files linked within the XML resource file will be converted and copied in a format usable by the *Internationalization API*. For this reason, you do not need to include them in your project source folders (actually, this is not recommended, since it would cause an increase in your Java app size).

The ResourceMaker tool

Once the XML resource file is ready and contains all the necessary resources, the *ResourceMaker* tool can be used to generate the resource files needed by the *Internationalization API*.

To start, the ResourceMaker ZIP file must be downloaded and extracted on your development machine.

After you've done that, *ResourceMaker* can be launched by browsing to the *JSR_238_RI_1_0\tools\ResourceMaker\bin* subfolder of the extracted ZIP file, and then using the following command-line syntax:

```
java com.nokia.phone.ri.global.tools.ResourceMaker <XML_RESOURCE_FILE_PATH>
<BINARY_RESOURCES_BASE_DIR> <GENERATED_RESOURCES_BASE_DIR> <RESOURCE_SET_BASE_NAME>
<LOCALES_LIST_SEPARATED_BY_SPACES>
```

By executing *ResourceMaker* as illustrated above, a folder named *global* gets generated in the <GENERATED_RESOURCES_BASE_DIR>, containing the following data:

- a file named **_<RESOURCE_SET_BASE_NAME>**, containing the list of locales specified on the *ResourceMaker* command line
- a folder for each locale, with each folder containing a file named **<RESOURCE_SET_BASE_NAME>.res**

Also, a Java class named `<RESOURCE_SET_BASE_NAME>.java` gets generated in the *<BINARY_RESOURCES_BASE_DIR>* folder. This class, together with the above files, will be used by the *Internationalization API* to properly localize the Java app, as shown in the following section.

Using the localized resources

Using the localized resources generated by the *ResourceMaker* tool requires the use of the ResourceManager class. A `ResourceManager` object can be instantiated using the static `getManager()` method, and specifying the desired **resource set base name** as argument, as shown in the following code snippet:

```
ResourceManager resManager;
try
{
    resManager = ResourceManager.getManager("MyBaseName");
}
catch(ResourceException e)
{
}
```


(C) Copyright Nokia 2014. All rights reserved.

The above call tries to instantiate a `ResourceManager` set on the default device locale. If this locale is not supported by the app resources, a ResourceException is raised. In this case, a `ResourceManager` can be instantiate by explicitly specifying a default locale (typically 'en'):

```
ResourceManager resManager;
try
{
    resManager = ResourceManager.getManager("MyAppLocale");
}
catch(ResourceException e)
{
    resManager = ResourceManager.getManager("MyAppLocale", "en");
}
```

Once instantiated, a `ResourceManager` is used to retrieve localized versions of textual and binary resources - based on the `name` values specified in the XML resource file. In order to use those constants, it is necessary to copy the **<RESOURCE_SET_BASE_NAME>.java** class, generated by the *ResourceMaker* tool, in your project sources.

Now it is possible to retrieve a textual resource:

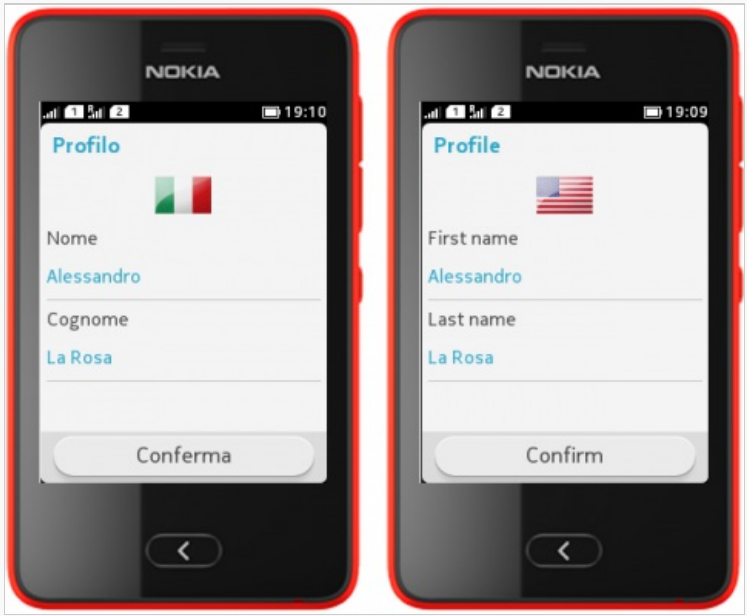
```
String textResource = resManager.getString(MyAppLocale.MY_RESOURCE_NAME);
```

Similarly, a binary resource can be retrieved as a `byte array`: this data can then be used to instantiate the appropriate Java object. The following code shows how an `Image` can be retrieved and instantiated:

```
byte[] imageData = resManager.getData(MyAppLocale.FORM_ICON);

Image iconImage = Image.createImage(imageData, 0, imageData.length);
```

The following picture shows a sample LCDUI `Form`, whose text and image resources are fully localized using the techniques illustrated above.



Localization of text and image resources

Formatting data

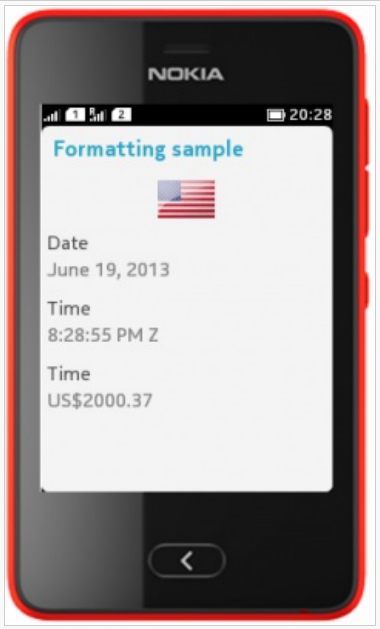
Static data, like text and images, is not the only that needs to be localized: depending on the locale, data like currency, time and dates also need proper internationalization and formatting. The *Internationalization API* brings full support for such functionality

(C) Copyright Nokia 2014. All rights reserved.

with its Formatter class. By using a `Formatter` instance, it's possible to apply proper formatting to various values, as follows:

- `formatCurrency()` formats currency values
- `formatDateTime()` formats date and time values
- `formatPercentage()` formats percentage values
- `formatNumber()` formats numbers

The following picture shows how the above methods correctly format various kind of values:



Date, time and currency formatted using the "en-US" locale

Formatting templates

The `Formatter` class also supports formatting of messages based on templates, with a template being a string containing place-holders that are replaced with custom values, determined at runtime.

The following code snippets show how this feature can be used to create and use fully localized template messages. Specifically, the following sample shows how to format a person's name in <FirstName> <LastName> or <LastName> <FirstName> format depending on the locale.

Localized template resources:

```
<?xml version="1.0"?>
<!DOCTYPE resources SYSTEM "resources.dtd">
<resources version="1.0">
    <set basename="MyAppLocale">

        <resource id="9" type="string" name="PROFILE_CONFIRM_TEMPLATE">
            <value locale="en">Your data is saved, thank you {0} {1}</value>
   <value locale="it-IT">Dati salvati, grazie {1} {0}</value>
        </resource>

    </set>
</resources>
```

Formatting of the localized template:

```
String template = resManager.getString(MyAppLocale.PROFILE_CONFIRM_TEMPLATE);

String message = Formatter.formatMessage(template, new String[]{firstName, lastName});
```


(C) Copyright Nokia 2014. All rights reserved.

The following picture shows how the above message gets properly localized and formatted depending on the selected locale.

Formatting of localized template messages

Locale-based string comparison

Another important feature of the *Internationalization API* is brought by the StringComparator, that allows to compare String objects taking into account the selected locale.

Localizing the app name

In order to complete the internationalization of a Java app, it is often necessary to localize its name in the various supported locales. This operation can be performed using Nokia-specific JAD manifest attributes, specifically:

- **Nokia-MIDlet-Name-<LOCALE>**: provides a localized version of the MIDlet suite name
- **Nokia-MIDlet-1-<LOCALE>**: provides a localized version of the MIDlet name

The following snippet, taken from a JAD file, shows how both MIDlet suite name and MIDlet name are localized in **en** and **it-IT** languages:

```
MIDlet-1: Asha Localization,,com.jappit.example.wikilocalization.WikiLocalizationMIDlet
Nokia-MIDlet-1-en: Asha Localization
Nokia-MIDlet-1-it-IT: Localizzazione Asha
MIDlet-Name: Example App
Nokia-MIDlet-Name-it-IT: App di Esempio
Nokia-MIDlet-Name-en: Example App
```

Those attributes are supported from Series 40 5th Edition Feature Pack 1 onwards.

The Locale API

The Locale API completes the internationalization feature set available on the Asha software platform. Supported since Nokia UI API 1.6, the **Locale API** is supported starting from the **Java Runtime 2.0.0 for Series 40**.

The features offered by the *Locale API* include:

- retrieving the Locale currently set on the device
- listing of the locales supported by the device
- registering of a LocaleListener object that gets notified when the device's locale changes

All those functionalities are exposed through the LocaleManager class.

The following code snippet and picture show how the locales available on a device can be listed using the `LocaleManager` object:

(C) Copyright Nokia 2014. All rights reserved.

```
LocaleManager localeManager = LocaleManager.getInstance();

Locale[] locales = localeManager.getAvailableLocales();

for(int i = 0; i < locales.length; i++)
{
    myList.append(locales[i].getLanguage() + "-" + locales[i].getCountry(), null);
}
```

List of the locales available on a Nokia Asha 501

Source code

The source code of a sample Java app, showing the techniques described in this article, can be downloaded here: Media:InternazionalizedMIDlet.zip

Summary

This article explores the features offered by the Nokia Asha software platform to allow the internationalization of a Java app, and specifically goes into the detail of the *Internationalization API*, the *Locale API* and *Nokia-specific features* to illustrate the various steps involved in the localization process.

More information is available on the Localisation pages of the Nokia Developer's Library.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.