Porting Android (Java) applications to Java ME on S60 5th Edition

Introduction

Android is a platform for mobile devices maintained by Google and supported by the Open Handset Alliance.

The Android platform includes an application framework, an operating system, a Java Virtual Machine called Dalvik, a web browser based on WebKit, SQLite database, middleware services and key applications.

The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. However, the Java APIs supported by Android are not standard Java ME APIs (JSRs), so in order to develop an Android application the developer must learn and cope with many new APIs.

Android is built on the Linux Kernel and utilizes a custom virtual machine, commonly referred as Dalvik VM. Although the Android's kernel is based on Linux, developers can only use Java programming syntax to develop Android applications.

Java Micro Edition applications are not compatible with Android due to the new format of byte codes used by Dalvik virtual machine and due to this fact developers must rewrite Android applications to work on S60 5th Edition devices or any other devices supporting standard Java ME.

This article provides an overview of Android in an extent that will allow you to port Android applications to S60 5th Edition, focusing on Java ME.

Android platform - Architecture

Android architecture is basically divided into several components: Application Framework, Libraries, Android Runtime and Linux Kernel.

(C) Copyright Nokia 2013. All rights reserved.

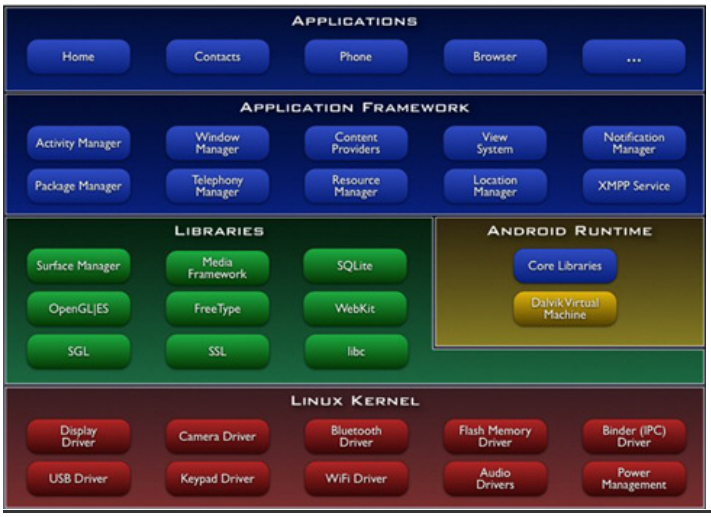
As mentioned before, Android has virtual machine called Dalvik and every Android application will run on its own process, that is, using its own instance of Dalvik JVM.

The Dalvik JVM runs files called Dalvik executables (.dex files), a kind of Java byte code optimized for the Android platform. The Linux Kernel is based on version 2.6.

At last, Android code can use the 1.5 version of Java language (Tiger), so the Java Tiger features are available for Android. It means you can use Generics, enhanced for loop, auto boxing e other new features of language when developing Android software.

Java RT for S60 has two newer versions, called JRT 1.3 (S60 5th Edition) and JRT 1.4 (S60 3rd Edition FP2). So, on the contrary, when porting Android code to S60 Java, in case you come across Java Tiger code, you will need to convert them back to Java 2 language, as both JRT 1.3 and 1.4 do not support Java Tiger.

For more information about JRT, please check this article Archived:Naming and versioning of Java Runtime for Symbian



Application Development

Application Components

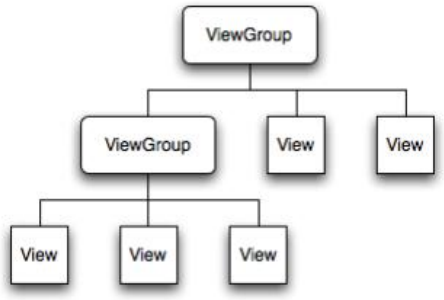
The main components of an Android application are: `Activity`, `IntentReceiver`, Service and `ContentProvider`. Depending on the kind of application one might be developing, not all components will be present in an application. In short, they are defined as below:

- An Activity has the main callback methods and is linked (represents) a blank screen in a logical way (an application may have several Activity classes);
- An Intent represents an action and it is composed by <action> and <category> XML elements. Then the IntentReceiver and IntentFilter (<intent-filter> element) will take care of its action processing and guarantee that the right Intent will be processed;
- AndroidManifest.xml - the components above are controlled by means of the AndroidManifest.xml configuration file, which is a kind of descriptor for the Android application.

User Interfaces

The main components are the **android.view.View** and **android.view.ViewGroup**.

- The View represents a graphical area on the device's screen and by means of it you can find out the width and height, set focus, control scrolling and key handling for the correspondent area. It is very extensible and several Android Widgets extend from View.
  - Some examples of Widgets are AbsListView, AbsSeekBar, AbsSpinner, AppWidgetHostView, AutoCompleteTextView, Button, CheckBox, CheckedTextView, Chronometer, CompoundButton, DatePicker, DialerFilter, DigitalClock, EditText, ExpandableListView, ExtractEditText, FrameLayout, GLSurfaceView, Gallery, GridView, HorizontalScrollView, ImageButton, ImageSwitcher, LinearLayout, ListView, MediaController, MultiAutoCompleteTextView, RadioButton, RadioGroup, RatingBar, RelativeLayout, ScrollView, SeekBar, SlidingDrawer, Spinner, TabHost, TabWidget, TableLayout, TableRow, TextSwitcher, TimePicker, ToggleButton, TwoLineListItem, VideoView, ViewAnimator, ViewFlipper, ViewSwitcher, WebView, ZoomButton and ZoomControls.
- The ViewGroup is a container of View, that is, it is a container class and helps defining the layout characteristics of your GUI.
  Some examples of layouts are AbsoluteLayout, FrameLayout, LinearLayout and RelativeLayout.



-

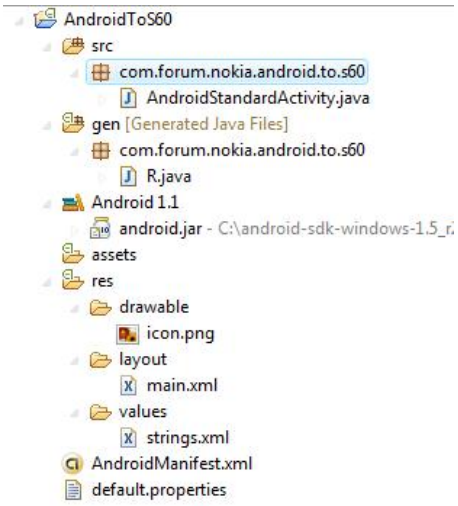
(C) Copyright Nokia 2013. All rights reserved.

Project Structure

XML files play a major role in android application development, mainly regarding configuration and declaration of resources. But standard Java ME does not use the same approach, so the mapping that is done using the **AndroidManifest.xml** (events mapping to actions) do not exist in Java ME by default. You will need to analyze the Android manifest and implement the logical mapping on your S60 Java ME application using well known patterns like MVC, Command, Dispatcher and others.

Just as an example, when developing an Android application, in order to create the GUI, navigation mapping and action linkage generally the developer uses some XML files and refer each component using its ID. In standard Java ME as available in S60 devices, there is no such mapping imposed by the platform so one should refer each component by name and instance reference, using its source file (class definition) itself.

Below we have a very basic project structure regarding an Android project:



In summary:

- The src folder is where your all the classes created explicitly by the developer will be located. You can see a basic Activity class in this folder, inside the com.forum.nokia.android.to.s60 package;
- The gen folder has several classes that are created automatically by Android and should never be edited manually by the developer;
- The android.jar that is included on the classpath and has the Android framework;
- The res folder that contains all the resources used by the application, such as images (drawable subfolder has icon.png file), layouts (layout subfolder has the main.xml file, defining the layout for GUIs), static values, kind of resource bundle (values subfolder has the string.xml file where declarative values will be declared);
- AndroidManifest.xml - as cited before, it is the main configuration file for the Android application.

Porting Guidelines

AndroidStandardActivity.java

Below we have the code for a simple Activity. Note that it overrides and implements the public void onCreate(Bundle savedInstanceState) method. This method must be overridden to allow an activity to be created.

In some way it can be compared to MIDlet's startApp() method, so while there is no direct matching of methods between Android and standard Java ME given that approaches and strategies are very different, when porting your application you will convert the main Activity (main screen) to your MIDlet's startApp() and then convert the other activities to classes that will implement the functionality on your Java ME application.

Depending on the coding style, architecture or convention you or your company use, you can convert them to:

- Java class that implements the Runnable interface (separate thread);
- Java class that extends Form or Canvas;
- Java class that extends Object but has several business methods (Business Object design pattern).

So, the takeaway here is that while Android works with several separate Activity classes, each having its own callback methods and others, your Java ME will have only one (or maybe more if preferred) MIDlet classes and the remaining Activity classes will be converted to one of the examples cited above (Thread, Canvas, Form or BO).

```
package com.forum.nokia.android.to.s60;

import android.app.Activity;
import android.os.Bundle;

public class AndroidStandardActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```


(C) Copyright Nokia 2013. All rights reserved.

R.java

The is no need to replicate the class below to standard Java ME application as it is proprietary to Android and Java ME does not consider the same approach. So you can ignore it, but it can help you figure out the correspondence that exists among layouts, strings and other resources in an Android application.

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.forum.nokia.android.to.s60;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

strings.xml

This file may be converted to standalone resource file (simple .txt file, for example, kind of resource bundle), JSR-238 file (MIA) or into entries on JAD file to hold your literal, constant values, avoiding some hardcoded values on your source code.

So there is no need to port this file, just create a similar one in S60 Java ME application to achieve the same purposes.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, AndroidStandardActivity!</string>
    <string name="app_name">AndroidToS60Porting</string>
</resources>
```

main.xml

This file expresses the layout being used by the Android application, regarding its GUI.

In Java ME, you can control it using the new layout features that are available in frameworks like LWUIT, eSWT or even the limited support that is available in MIDP. So there is no need to port this file or create a similar one in S60 Java ME application.

To achieve the same layout organization, it is advisable to read and understand the layout classes that are available in Android, cited above in one of the previous sections of this article.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

Java APIs (packages) - Android and Java ME

To get the detailed description about each class you can refer the Javadoc documentation that is available in both platforms.

- S60 5th Edition - Java ME
- Android

Deployment Packages

Nokia - JAD, JAR files

Android - APK files

Application Signing

Nokia supports: VeriSign, Thawte, Java Verifed (UTI) certificates

 (C) Copyright Nokia 2013. All rights reserved.

Android supports: VeriSign, Thawte, self-issued certificates (using tools like keytool)

Tools - SDKs and IDEs

- Wireless tool kit(WTK)
- Nokia S60 SDK
- NetBeans
- Eclipse

Restricted APIs (permissions)

On Android the developer needs to set permissions in the AndroidManifest.xml file to allow the access to some restricted APIs or resources. Android applications must also be signed with a certificate whose private key is held by their developer, but it can use a self-issued certificate.

Regarding Java ME, if application is not signed then the application will ask the permission to user while executing it and by signing it will be granted automatically, so there is no need to get explicit permission confirmation for most of the resources and restricted API calls.

Data Storage

J2ME : RMS, FileConnection

Android : Preferences, Files, Databases, Network storage and SQLite database

(C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.