Launching Java apps at phone startup

This article introduces the **Java Launcher** package, a tool that allows developers to launch their Java ME applications at phone startup/installation on Symbian devices.

05 Sep 2010

Prerequisites

In order to customize this package to your own needs, you need to have installed:

- Some S60 SDK, 3.0 MR or greater. This example was created with S60 5th Edition SDK.
- Carbide.c++ 2.0+ is optional but helps getting the job done more quickly.
- **Java Launcher** project in Zip format: File:JavaLauncher 18122009.zip
- **ESWT Showcase** Java MIDlet example: ESWT Showcase.zip

Testing this package

First thing is to do a test build to make sure the environment is set up correctly:

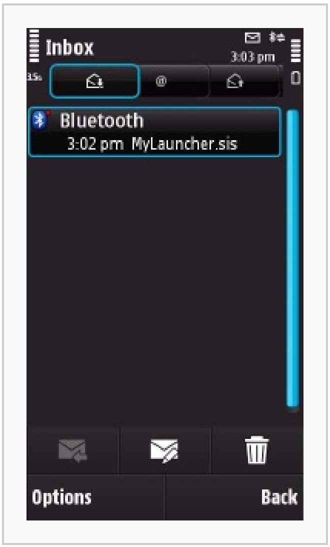
1. Install the chosen SDK, if you still haven't done so.
2. Download and install the example Java application that will be executed in this test: <insert file here>
3. Extract the attached **JavaLauncher_18122009.zip** package anywhere in your computer. In this example, the folder generated is **C:\temp\JavaLauncher**
4. Open a Windows Command Prompt, and cd into the project folder.
5. Cd into **group** folder and type `bldmake bldfiles` to create the build files.
6. Type `abld build gcce urel`. This will build the package for the GCCE target, meant to be used in real hardware.
7. Cd into **..\sis** folder.
8. Type `makesis JavaLauncher_EKA2.pkg JavaLauncher.sis`. This will generate an unsigned .sis file. You will now need to sign the file so it works on your phone.

Obtain a Developer certificate from the Symbian Signed website and:

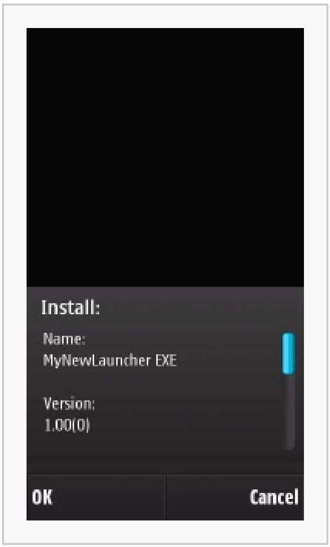
1. Sign **JavaLauncher.sis** with your certificate and key.

Now you can install the signed **JavaLauncher.sis** file on your phone. The Symbian installer will prompt you about the file being a development-version application, then will install itself. After the installation of the main package is completed, the installer will try to execute the ESWT Showcase application you downloaded and installed earlier. If the launch fails, it will display an error message and exit. Otherwise, the ESWT Showcase will be executed. You can now reboot your phone and verify that ESWT Showcase will be launched after the system has started up, which is exactly what we wanted. Here is a visual summary of the installation experience on the phone:

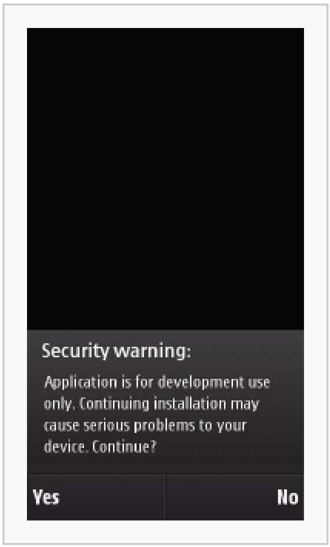
Installation of multiple apps with a single .sis file



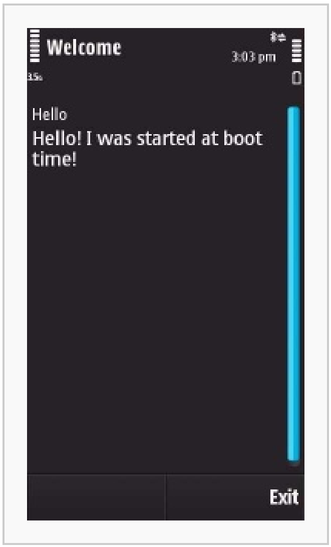
MyLauncher.sis in the Inbox


Install:
Name:
MyNewLauncher EXE
Version:
1.00(0)

Installation of the main package


Security warning:
Application is for development use only. Continuing installation may cause serious problems to your device. Continue?

Signing with a developer certificate prompts the user



Hello
Hello! I was started at boot time!

"Hello world!" executed after installation and on each reboot

Next we'll see how you can customize this package to suit your needs.

Customizing the launcher package

The first thing you need to know is that this is not simply a .sis file. There's a small executable ("**JavaLauncher.exe**") that does the job of finding and executing the correct Java application. This has implications in what you, the developer, must do to customize this package.

In the Symbian environment, all applications must have a valid, unique ID (**UID**) and a unique executable file name, to avoid collision with other applications installed on the phone. Therefore you need to implement those changes before using this package, in order to install your applications in a production environment. To achieve this, you need to:

1. Go to the Symbian Signed website, and get a valid UID for your application. It will be in the format 0x12345678. Please notice that the example package comes with a UID of the test range (0xExxxxxxx) which must not be used for production applications. You will need a valid one or you will have problems with application collision and it will be impossible to sign them.
2. After you have the valid UID, open the **JavaLauncher.mmp** (**group** folder) file and change the line

   ```
   UID    0 0xEDC63187
   ```

   to

   ```
   UID    0 0x2xxxxxxx
   ```

   where **0x2xxxxxxx** is the UID you've received from Symbian Signed. Modify the **TARGET** parameter to something like **MyApplicationName_0x2xxxxxxx.exe**, again where **0x2xxxxxxx** is your UID. This will ensure the file name for your executable is unique in the phone. Also look for the line

   ```
   START RESOURCE ..\DATA\E9F13F63.rss
   ```

   and change it to

   ```
   START RESOURCE ..\DATA\2xxxxxxx.rss
   ```

   . Save and close the file.
3. CD into the **data** folder and rename the **E9F13F63.rss** file to **2xxxxxxx.rss**. The file name must match the application UID but without the leading "0x". Open the .rss file and change the line

   ```
   executable_name = "!:\\sys\\bin\\JavaLauncher.exe";
   ```

   to

   ```
   executable_name = "!:\\sys\\bin\\MyApplicationName_0x2xxxxxxx.exe";
   ```

   which is just the same application .exe file name you entered in the .mmp file. Save this file and close it.
4. Open the **..\sis\JavaLauncher_EKA2.pkg** file. Look for the line:

   ```
   #{"JavaLauncher EXE"},(0xEDC63187),1,0,0
   ```

   and change it to

   ```
   #{"MyApplicationNAme"},(0x2xxxxxxx),1,0,0
   ```

   where **MyApplicationName** is, unsurprisingly, your application's name, and **0x2xxxxxxx** is the same UID you used in the

.mmp file. You can also change the **Vendor** section to reflect your company's name.

5. Look for the line

```
"\S60\devices\S60_5th_Edition_SDK_v1.0\Epoc32\release\gcce\urel\JavaLauncher.exe"
- "!:\sys\bin\JavaLauncher.exe", FR, RI
```

   and change the name of **JavaLauncher.exe** in both sides to the same name you've defined in step 2 of this section ("**MyApplicationName_0x2xxxxxxx.exe**"). Keep the file open.
6. Look for the line

```
"\S60\devices\S60_5th_Edition_SDK_v1.0\Epoc32\data\E9F13F63.rsc" -
"c:\private\101f875a\import\[E9F13F63].rsc"
```

   and change the name of **E9F13F63.rsc** on the left side to **2xxxxxxx.rsc**, and on the right side to **[2xxxxxxx].rsc**. This is the compiled resource filed generated from <YOURUID>.rss file in the **data** folder. Don't forget the square brackets ([ ]) on the *right side*, they are needed by the system.
7. Edit **..\src\JavaLauncher.cpp** (Notepad or Wordpad) change the line

```
_LIT(KMidletToLaunch, "eSWTShowcase");
```

   to

```
_LIT(KMidletToLaunch, "YOUR_MIDLET_NAME");
```

   where "YOUR_MIDLET_NAME" must match the MIDlet name defined in the JAD or Manifest.mf files in the MIDlet suite. The MIDlet in these name is the first value of the 'MIDlet-1' attribute. In the example below, the MIDlet name is "HelloMIDlet":

```
MIDlet-1: HelloMIDlet, , hello.HelloMIDlet
```

   Save and close the file.
   ATTENTION: It's very important to note that the MIDlet needs to be installed on the device before the launcher package in order to be successfully started. In the next section we will see how to automate the entire process by installing the Launcher and the MIDlet as a single .sis package.
8. Now you are ready to rebuild the package. Change to the **group** folder, type

```
bldmake bldfiles
```

   then

```
abld reallyclean gcce
```

then

```
abld build gcce urel
```

1. Create the package using **makesis** as described in the previous section.
2. Sign the package using either Symbian Signed Open Signed Online or your developer certificate.
3. You are then ready to test the package on your phone.
4. **ATTENTION**: At this point you may receive an "Update error" when installing your newly-created package on the phone.

This happens because you've changed the UID of the package but kept some of the files that the old package (the one installed on the phone) was also using. To solve this error, you can either remove the previous package prior to installing the new one, or remove conflicting files, which will most likely be the ones from the **data** folder. This will not happen in production since it's very unlikely that you are installing exactly the same file as any other package in the world with a different UID.

Deploying the installation package

In order to deploy the installation package to all your users, you need to sign the application with a real certificate. For that you will need to go to Symbian Signed and sign the application. You can use Express Signed since the app doesn't require any Manufacturer-granted capabilities. For an overview of the Symbian Signed process for production applications Step by Step instructions to Express signing.

Installing everything as a single package

As you have noticed, in order for the launcher package the work, the target MIDlet needs to be installed on the device already. This is not very convenient as you'd have to tell your users to install two different packages, which is not a good experience. In order to solve this problem, you can use the Software Installer package to install the launcher package and the Java MIDlet as a single file, providing a good experience for your users.

To get that done, you just need to follow the instructions in the Software Installer article on how to customize the package, and ensure that the Java MIDlet is installed before the boot launcher.

Below there is an example package showing how it should be done. The "MyLauncher.sis" file used in the example is the one generated in this article.

File:SingleShot.zip

Questions / feedback

Questions or feedback can be posted as comments to this page.