OpenJFX on Freescale i.MX6

JavaFX is known to work on i.MX6 platforms with 'hard float ABI" Linux distributions.

Each vendor usually supplies a sample distro that has been configured to work properly with the i.MX6 board they provide. See the notes below for known boards, and please help us by adding boards you have tested that are not present.

Currently we recommend Boundary Devices. They have made a special effort to support switching to accelerated framebuffer mode, and have a vibrant community that answer device questions quickly.

Please note, there are two environments for running JavaFX that are regularly tested.

- Direct to Frame buffer (no X11). This is the targeted path for JavaFX on ARM and the most tested
- Using X11 to obtain the EGL context, but acting as if we are the only application. This path was implemented in the "Monocle" windowing subsystem, which is available in the OpenJFX build, and will be part of a future JDK for ARM release. This path was added to enable us to demonstrate JavaFX on platforms where the direct to Framebuffer drivers are not easily available. There are limitations inherent to using this path, and so it is not recommenced for production use.

Each of these paths require a different set of EGL libraries, due to choices made by the hardware vendor.

Start with JDK 8 for ARM. Unpack this binary into a known location like `/opt`.

If you want to use the latest OpenJFX (and Monocle) then you will need to build it, and copy the resulting binary overlay on top of the JDK 8 for ARM.

OpenGL Accelerated Libraries (EGL)

The Vivante framebuffer specific divers must be present and properly configured. The EGL drivers are built to support a particular rendering target (X11, framebuffer, ...). This affects libEGL.so, libGAL.so and in some cases libVIVANTE.so. Sometimes the "alternate" version of the library is present in /usr/lib, as libEGL-fb.so for example. This alternate version must either be:

- copied to an alternate location (without the -fb) and used with LD_LIBRARY_PATH
- used as the target for the /usr/lib/libEGL.so symbolic link

The kernel also needs the proper drivers to be present for these libraries to work. This can be checked with:

```
$ dmesg | grep Galcore
Galcore version 4.6.9.9754
```

UDEV

Udev must be configured on the target system. JavaFX tries to open libudev.so.0

Run as root

Embedded JavaFX used udev and direct access to the framebuffer (/dev/fb0). In general, JFX Embedded applications will need to be run as root, unless the system has been reconfigured to grant access to these devices.

Yocto

Currently Yocto is supported on many i.MX6 based devices. The documented and tested mainline target (fsl-image-x11) builds for an X11 based image. There is work underway for a framebuffer based target (fsl-image-fb), but that is not currently on the Yocto mainline.

A framebuffer only image will need fonts installed and configured.

Note: currently there is an false dependency on libX11.so in the JDK8 preview. See the JIRA for more details and possible workarounds.

In a Yocto fsl-image-gui build, the framebuffer versions of the needed libraries are present in the build tree. Look for libEGL-fb.so and (libGAL-fb.so, libVivante-fb.so). Copy this to a directory in your working image removing the "-fb".

Boundary Devices

Boundary Devices has produced a very easy to use image based on Ubuntu which can be obtained from here.

The SHA1 sum is 6b38a1eda87fb43e5b5f3587e760c6489ddd2d0f.

```
You can restore from Linux using zcat and dd:

        ~/Downloads$ sudo umount /dev/sdx?
        ~/Downloads$ zcat vpu*.gz | sudo dd of=/dev/sdx bs=1M
        ~/Downloads$ sync
```

Or on a Windows machine, you can use Alex Page's USB Image Creator: http://alexpage.de/usb-image-tool/

When you boot the device, you should be logged in as user 'ubuntu', password 'Boundary'.

Now unpack your JDK 8 for ARM into /opt. It is helpful to create a symbolic link to shorten the path:

```
ubuntu@oracle:/opt$ sudo ln -s jdk1.8.0_06 jdk
```

If using OpenJFX, copy the build results over the top of the of the installed JDK 8 for ARM.

By default, the EGL libraries are targeted for X11. If you have OpenJFX installed, you should be able to run a JavaFX application and it will use X11, but in a simulated full screen mode. Things to check, especially if running from SSH:

- Do you have your DISPLAY variable set to :0.0 ?
- Can you launch a simple X11 app (like xclock) and does it show up on the correct display ? If you cannot launch a simple native X11 application, then you will want to resolve those issues first (usually xauth related).

To run the preferred direct to Framebuffer path, the packaging allows you to switch to the frame-buffer versions of the Vivante libraries using 'apt-get'.

```
root@oracle:~# stop lightdm
root@oracle:~# apt-get install gpu-viv-acc-fb
```

When running a JavaFX application in this mode, **you will need to run using sudo**, as JavaFX needs to open up a number of devices that read/write only for root.

```
ubuntu@oracle:~# sudo /opt/jdk/bin/java -cp ...
```

To switch back to the X11 accelerated EGL libraries, use:

```
root@oracle:~# apt-get install gpu-viv-acc-x11
root@oracle:~# start lightdm
```

Yocto

The Yocto demonstration build has been show to work, with a bit of effort. The challenge is to obtain a "framebuffer" version of of the graphics libraries (libEGL.so,libGAL.so). Once you have those, stop X11, and use LD_LIBRARY_PATH to point to the overriding libraries.

```
/etc/init.d/xserver-nodm stop

export LD_LIBRARY_PATH="path to the framebuffer versions"

export PATH=$PATH:/opt/jdk1.8.0/bin
```

The above assumes you don't want to break the default X11 configuration quite yet.

(TODO find a proper link to the framebuffer versions of libEGL.so,libGAL.so)

Raring

The Raring build is known to work, though the cursor is not rendered properly. Note in this image, that you need to use all three versions of the framebuffer libraries (libEGL.so,libGAL.so, libVIVANTE.so)