Cross Building for ARM Hard Float

Cross building for ARM hard float

Just Looking for a binary to try? Check out the Community builds.

In order to cross-compile for Linux/ARM you first need to obtain ARM based libraries to compile against. You'll need several of the same tools as for a Linux desktop build, so you should start by making sure you can build for the desktop, following the instructions above.

There are three general parts to a cross compile tool chain, and there is a script in OpenJFX that will gather these items, and put them in the appropriate place to match the build configuration. These parts include:

- A cross compiler
- Include files
- target libraries to link against

Note that the target libraries often do not have to match exactly the system they will run on, as long as the version number matches what is available on the target system. This means we can use a 'generic' version of most shared libraries, and still be able to work on the target system. There are certain libraries that are not generic - libEGL often will have vendor specific dependencies for example, making the library non-portable to link against.

A Linux command 'ldd -r a_shared_library.so' can be used to check a shared library on the target system, and validate that any dependencies are met. In particular this command can be used to test the OpenJFX native libraries to verify they are compatible.

Fetching a cross compile toolchain

These instructions assume that you don't need to set an HTTP or HTTPS proxy to access the internet, either because you don't need one or because your system is already configured to use one. If you need to define proxy settings then you should define the environment variables `http_proxy` and `https_proxy`. Both are needed. For example,

```
export http_proxy="http://<proxy-host>:<proxy-port"
export https_proxy="http://<proxy-host>:<proxy-port"
```

If you are installing a package using sudo, you need to define the HTTP proxy after sudo, like this:

```
sudo http_proxy="http://<proxy-host>:<proxy-port" apt-get...
```

To obtain the toolchain, in the OpenJFX directory us the following commands to assemble a toolchain. Note: this shell script relies of a Debian base distro, so will not work on Fedora.

```
mkdir tmp
cd tmp
../buildSrc/crosslibs/crosslibs-armv6hf.sh
# check to see that ../../crosslibs is present
ls ../../crosslibs
#and if it is
cd ..
rmdir tmp
```

This will download Debian packages and unpack them into a directory `crosslibs` at the same level as your OpenJFX working copy. It will then download a cross-compiler for ARM and install it in the same place. When the script has completed you should see:

```
Done.
```

You are now ready to run a full cross-compile for ARM hard float.

Note: the script assumes a Debian based distro (like Ubuntu), it will not run on Fedora.

Note: if you are using a 64bit Linux, compatibility libraries will need to be installed to support the 32bit based compiler. This would require:

```
# Ubuntu 12
sudo apt-get install \
    ia32-libs \
    zlib1g-dev:i386

#Ubuntu 14, 15
sudo apt-get install \
    gcc-multilib \
    g++-multilib \
    zlib1g-dev:i386

#Fedora 21
yum install glibc.i686 libgcc.i686 libstdc++.i686 glibc-devel.i686 zlib.i686
```

The compile command is run on the Linux x86 machine:

```
gradle -PCOMPILE_TARGETS=armv6hf
```

This command tells Gradle to use the configuration file buildSrc/armv6hf.gradle for this build. There are several flavors of ARM Linux, the most common and most tested is armv6hf (ARM v6, hard float ABI).
The results of this build will be located in ./build/armv6hf-sdk. A Linux desktop build is in ./build/sdk. This means that you may build for both in the same repository. To build for both desktop, and ARM at the same time, you can add to the targets like this -PCOMPILE_TARGETS=linux,armv6hf

Testing a resulting build

When you run with this build you will need to tell Java to use your JavaFX instead of its built-in JavaFX. For example,

```
sudo /opt/jdk1.8.0/bin/java \
    -Djava.ext.dirs=build/armv6hf-sdk/rt/lib/ext \
    -jar BrickBreaker.jar
```

Note that setting java.ext.dirs overrides the location of the JRE extension directory, and so any other jars present in the extension directory of your JRE will not be seen.

As an alternative to java.ext.dirs, you can copy the build result on top of a copy of your JRE installation

```
JAVA_HOME=/opt/jdk1.8.0 # replace with your installation path
# remove some older JFX bits so they don't collide
rm -f $JAVA_HOME/jre/lib/ext/jfx*jar $JAVA_HOME/jre/lib/arm/libjavafx_font_t2k.so
# and then overlay the built bits
cp -r build/armv6hf-sdk/rt/lib $JAVA_HOME/jre/
```

Note that JavaFX on ARM assumes that it is running as root so that it may open the framebuffer and read directly from udev. It is certainly possible to set up system permissions so that running as root is not required.