Sailfish OS Hardware Adaptation Development Kit Documentation

Release 3.0.0.0

Jolla Ltd.

Dec 17, 2018

CONTENTS

This is a guide to help you understand how you can port Sailfish OS to devices running the CyanogenMod flavour of Android.

> **Warning:** Modifying or replacing your device's software may void your device's warranty, lead to data loss, hair loss, financial loss, privacy loss, security breaches, or other damage, and therefore must be done entirely at your own risk. No one affiliated with this project is responsible for your actions but yourself. Good luck.

CHAPTER

ONE

OVERVIEW

1.1 Goal

By following this guide you can set up a Sailfish OS (or another Mer Core based) Linux system that will run on an Android device, on top of an existing Android Hardware Adaptation kernel and drivers.

This consists of:

- **Sailfish OS core (Mer)**: the GNU/Linux userspace core
- **Android Hardware Adaptation** (HA/HAL), consisting of:
  - Device-specific **Android Kernel**
  - **Binary device drivers** taken from an Android ROM (e.g. CyanogenMod)
  - The **libhybris interface** built against the binary drivers
  - **Middleware packages** depending on hardware-specific plugins
  - A Qt/Wayland **QPA plugin** utilizing the Android `hwcomposer`
- **Sailfish OS** components

1.2 Development

1.2.1 Requirements

The development environment uses the Platform SDK, with:

- one or more device specific **targets** (a rootfs with device-specific headers and libraries)
- a HA build SDK (a minimal Ubuntu chroot required to build the Android sources)

During the HA development you'll typically have one window/terminal using the HA build SDK where you build and work on Android code and another session using the Platform SDK where you build RPMs for the hardware adaptation.

Setting up the Platform SDK, as well as the device-specific targets and the Ubuntu HA build chroot is described in *Setting up the SDKs*.

Commands and output from the Platform SDK session are indicated using `PLATFORM_SDK $` at the top of the code block, like this:

```
PLATFORM_SDK $

echo "run this command in the Platform SDK terminal"
```

How to enter `PLATFORM_SDK $` is explained in *Setup the Platform SDK*.

Commands and output from the HA build session are indicated using `HABUILD_SDK $` at the top of the code block, like this:

```
HABUILD_SDK $

echo "run this command in the Ubuntu HA build SDK terminal"
```

How to enter `HABUILD_SDK $` is explained in *Entering Ubuntu Chroot*.

1.2.2 The build area root directory

In this guide, we refer to the SDK directory hosting Platform SDK, Targets, and Ubuntu chroot with the environment variable `$PLATFORM_SDK_ROOT`. With one SDK target spanning 0.5-1GB, you need around 3GB of space in total.

1.2.3 Build components

There are a number of components to build; the lower level and Android related components are built in the HA build SDK; the rest are built in the Platform SDK.

- In the **HA build SDK**
  - a kernel
  - a hacking friendly initrd which supports various boot options
  - `hybris-boot.img` and `hybris-recovery.img` (for booting and debugging)
  - a minimal Android `/system/` tree
  - modified Android parts for compatibility with libhybris and Sailfish OS (e.g. Bionic libc, `logcat`, `init`, ...)
- In the **Platform SDK**
  - RPM packages containing all the built binaries and extracted configs
  - Hardware-specific middleware and plugins (e.g. Qt QPA plugins, PulseAudio)

For distribution, RPM packages are uploaded to a HA-specific repository. With this repository, full system images using the `mic` utility. The `mic` utility is usually also run inside the Platform SDK.

1.3 Deployment

The `hybris-boot.img` (containing both the kernel and our custom initrd) is flashed to the device, while the Sailfish OS rootfs is placed in a subdirectory of the `/data/` partition alongside an existing, unmodified Android system.

The Sailfish OS rootfs is then used as a switchroot target with /data bind-mounted inside it for shared access to any user data.

CHAPTER

TWO

PREREQUISITES

2.1 Mobile Device

- An ARM Android device officially supported by CyanogenMod 10.1.x, 11.0, 12.1, 13.0, 14.1 (at the time of writing 2018-06-27). Also check this link
  - We are aware that CyanogenMod is made obsolete, and whilst going through this guide, please in your mind substitute the word CyanogenMod with an appropriate base that you are porting on (LineageOS, AOSP, CAF etc)
  - Starting with CM 13.0 (Android 6), support for 64bit ARM is also being added to Sailfish OS: firstly by running a mix of 64bit Linux Kernel and Android HAL, whilst Sailfish OS userspace is being run in 32bit mode
  - We will gradually transition from CyanogenMod to Lineage OS, in the meantime please use https://archive.org for CM and the mirror for their ZIP downloads: https://www.reddit.com/r/Android/comments/5kfm8x/the_cyanogenmod_archives_full_downloads
  - See http://wiki.lineageos.org/devices.html for a list of compatible devices, as well as the exhaustive list: http://web.archive.org/web/20161225121104/https://wiki.cyanogenmod.org/w/Devices
  - See https://wiki.merproject.org/wiki/Adaptations/libhybris for a status list of devices already ported using HADK
  - See https://wiki.merproject.org/wiki/Adaptations/libhybris/porters for a list of ports in early stages, and their authors to contact on the IRC
  - AOSP or CAF Android base support is also possible, but we chose CM/LineageOS for a wider range of devices. It will be up to the porter to patch an AOSP/CAF base with hybris patches. Remaining differences in using it are minimal (e.g. using the `lunch` command instead of `breakfast`)
- Means to do backup and restore of the device contents (e.g. SD card or USB cable to host computer), as well as flash recovery images to the device

2.2 Build Machine

- A 64-bit x86 machine with a 64-bit Linux kernel
- Sailfish OS Platform SDK (installation explained later)
- Sailfish OS Target (explained later)
- At least 16 GiB of free disk space (10 GiB source download + more for building) for a complete Android build; a minimal download and HADK build (only hardware adaptation-related components) requires slightly less space
- At least 4 GiB of RAM (the more the better)

CHAPTER

THREE

PREPARING YOUR DEVICE

Verify that you can backup and restore your device and that you understand device recovery options. This is not only useful when flashing images you build with this guide, but also in case you want to reset your device to its factory state with stock Android (note that not all Android vendors provide factory images for download, so you might need to create a full backup of your running Android system and store it in a safe place before starting to erase and reflash the device with your custom builds).

3.1 Backup and Verify Your Device

As mentioned above, it might be helpful to backup the stock image before flashing the CM release for the first time, as getting the stock image might be hard for some vendors (e.g. some stock images are only available as self-extracting .exe package for Windows) or impossible (some vendors do not provide stock images for download).

Use an Android/CyanogenMod Recovery to:

1. Backup to SD card: system, data, boot and recovery partitions
2. Test restoring the backup (important)

> **Warning:** While backing up to internal device storage is possible for some devices, if during porting you end up overwriting that partition, your backups will be gone. In that case (and in case of devices without SD card slots), it's better to also copy the backup data to your development machine (e.g. via `adb pull` in recovery). Recent versions of `adb` support full-device backups to a host computer using the `adb backup` feature.

See the ClockworkMod Instructions for additional help.

3.2 Flash and Test CyanogenMod

The official CyanogenMod flashing instructions can be found on this CyanogenMod wiki page.

You may also want to verify that the CM build for your device is fully functional, to avoid wasting time with hardware adaptations that have known issues. Also, your device might have some hardware defects - testing in Android verifies that all components are working correctly, so you have a functionality baseline to compare your build results with.

You should at least check the following features:

- **OpenGL ES 2.0**: Use e.g. Gears for Android to test (the hz you will get there will be max refresh rate).
- **WLAN connectivity**: Connect to an AP, ad-hoc or set up a mobile access point with your device.
- **Audio**: Headset detection, earpiece speaker, loudspeakers, etc.
- **Bluetooth**: Connect to bluetooth headsets, verify discoverability, send files.
- **NFC**: Check if NFC tags can be detected, read and/or written by the device.

- **SD/MicroSD**: Use a file manager app to see if inserted SD cards can be detected.
- **USB**: MTP, mass storage (if available) and `adb` access.
- **Telephony**: 2G/3G/LTE calls + data connectivity.
- **GPS**: Using GPS Test, check GLONASS too; typical time to fix; AGPS.
- **Sensors**: Using AndroSensor: Accelerometer, Proximity Sensor, Ambient Light Sensor, Gyroscope, Magnetometer (Compass), Hall (flip case), . . .
- **LEDs**: If your device has notification LEDs or keypad backlights.
- **Camera** (front and back): Also test functionality of zoom, flash, etc..
- **Buttons**: Volume up, volume down, power, camera shutter, etc..
- **Video out**: HDMI / MHL connectivity if you have the necessary adapters. TV out.
- **Screen backlight**: Suspend and backlight control, minimum and maximum brightness.
- **Battery meter**: Charge level, battery health, charging via USB (wall charger and host PC).
- **Vibration motor**: Intensity, patterns.
- **HW composer version**: check `dumpsys SurfaceFlinger` through ADB (see SF Layer Debugging).
- **Fingerprint sensor**
- **FM Radio**

We recommend that you write down the results of these tests, so you can always remember them.

CHAPTER

FOUR

SETTING UP THE SDKS

4.1 Setting up required environment variables

Throughout this guide we will be referencing the location of your SDK, targets and source code. As is customary with Android hardware adaptations, the device vendor (`$VENDOR`) and device codename (`$DEVICE`) are also used, both in scripts and configuration files. **Throughout this guide as example, we'll use Nexus 5 (lge/hammerhead for its vendor/device pair), and port it basing on CyanogenMod 11.0 version.** Thus ensure you read snippets carefully and rename where appropriate for your ported device/vendor/base.

Now run the following commands on your host operating system fitting for your device and setup:

```
HOST $

cat <<'EOF' > $HOME/.hadk.env
export PLATFORM_SDK_ROOT="/srv/mer"
export ANDROID_ROOT="$HOME/hadk"
export VENDOR="lge"
export DEVICE="hammerhead"
# ARCH conflicts with kernel build
export PORT_ARCH="armv7hl"
EOF

cat <<'EOF' >> $HOME/.mersdkubu.profile
function hadk() { source $HOME/.hadk.env; echo "Env setup for $DEVICE"; }
export PS1="HABUILD_SDK [\${DEVICE}] $PS1"
hadk
EOF
```

This ensures that the environment is setup correctly when you use the `ubu-chroot` command to enter the Android SDK.

It also creates a function `hadk` that you can use to set or reset the environment variables.

4.2 Setup the Platform SDK

Instructions are found on Sailfish OS wiki ("Quick start" section is enough, do not install SDK Targets yet): https://sailfishos.org/wiki/Platform_SDK_Installation

Afterwards, topup the newly created `~/.mersdk.profile` with necessary commands:

```
HOST $

cat <<'EOF' >> $HOME/.mersdk.profile
function hadk() { source $HOME/.hadk.env; echo "Env setup for $DEVICE"; }
hadk
EOF
```

You'll need some tools which are not installed into the Platform SDK by default:

- **android-tools-hadk** contains tools and utilities needed for working with the Android SDK
- **tar** is needed to extract the ubu-chroot image

```
PLATFORM_SDK $

sudo zypper ref
sudo zypper in android-tools-hadk tar
```

We strongly encourage all porters to use at least 3.0.0.8 Platform SDK. Use `sdk-manage` command to upgrade your toolings and targets, or create from new (especially when updating from 2.x to 3.x). To check what release you are on:

```
PLATFORM_SDK $

# if no such file, you're on an old SDK version
cat /etc/os-release
```

4.3 Setting up an Android Build Environment

4.3.1 Downloading and Unpacking Ubuntu Chroot

In order to maintain build stability, we use a *Ubuntu GNU/Linux* `chroot` environment from within the Platform SDK to build our Android source tree. The following commands download and unpack the rootfs to the appropriate location:

```
PLATFORM_SDK $

TARBALL=ubuntu-trusty-20180613-android-rootfs.tar.bz2
curl -O https://releases.sailfishos.org/ubu/$TARBALL
UBUNTU_CHROOT=$PLATFORM_SDK_ROOT/sdks/ubuntu
sudo mkdir -p $UBUNTU_CHROOT
sudo tar --numeric-owner -xjf $TARBALL -C $UBUNTU_CHROOT
```

4.3.2 Entering Ubuntu Chroot

```
PLATFORM_SDK $

ubu-chroot -r $PLATFORM_SDK_ROOT/sdks/ubuntu

# FIXME: Hostname resolution might fail. This error can be ignored.
# Can be fixed manually by adding the hostname to /etc/hosts
```


CHAPTER

FIVE

BUILDING THE ANDROID HAL

5.1 Checking out CyanogenMod Source

Our build process is based around the CyanogenMod projects source tree, but when required we've modified some projects, in order to apply patches required to make libhybris function correctly, and to minimise the built-in actions and services in the `init.*.rc` files.

Ensure you have setup your name and e-mail address in your Git configuration:

```
PLATFORM_SDK $

git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

You also need to install the `repo` command from the AOSP source code repositories, see Installing repo.

After you've installed the `repo` command, a set of commands below download the required projects for building the modified parts of Android used in hybris hardware adaptations.

All available CM versions that you can port on can be seen here: https://github.com/mer-hybris/android/branches

Choose a CM version which has the best hardware support for your device.

The result of your Sailfish OS port will be an installable ZIP file. Before deploying it onto your device, you'll have to flash a corresponding version of CyanogenMod, so Sailfish OS can re-use its Android HAL shared objects.

If your primary ROM is not CyanogenMod, or is of another version, look for MultiROM support for your device. It supports Sailfish OS starting v28.

```
HABUILD_SDK $

sudo mkdir -p $ANDROID_ROOT
sudo chown -R $USER $ANDROID_ROOT
cd $ANDROID_ROOT
repo init -u git://github.com/mer-hybris/android.git -b hybris-11.0
```

5.2 Device repos

The local manifest contains device-specific repositories, for Android as well as for the mer-hybris builds.

If your device has already been ported, its codes properly placed on GitHub, you should check this repository: https://github.com/mer-hybris/local_manifests (choose the branch of hybris-* that your are porting to), and use $DEVICE.xml file instead of creating a new one in this chapter.

Create directory at first:

```
HABUILD_SDK $

mkdir $ANDROID_ROOT/.repo/local_manifests
```

If your are working on a new port, you'll have to create the local manifest yourself, which contains at least two repos: one for the kernel, another for the device configuration. Find those CM device wiki, for Nexus 5 it would be http://wiki.cyanogenmod.org/w/Hammerhead_Info inside the **Source code** table. Local manifest below will also need pointing to correct branches - identify which one matches the default manifest branch (`stable/cm-11.0` in Nexus 5 case).

Add the following content to `$ANDROID_ROOT/.repo/local_manifests/$DEVICE.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <project path="device/lge/hammerhead"
    name="CyanogenMod/android_device_lge_hammerhead"
    revision="stable/cm-11.0" />
  <project path="kernel/lge/hammerhead"
    name="CyanogenMod/android_kernel_lge_hammerhead"
    revision="stable/cm-11.0" />
</manifest>
```

Time to sync the whole source code, this might take a while:

```
HABUILD_SDK $

repo sync --fetch-submodules
```

The expected disk usage for the source tree after the sync is **13 GB** (as of 2015-09-09, hybris-11.0 branch). Depending on your connection, this might take some time. In the mean time, make yourself familiar with the rest of this guide.

5.3 Configure Mountpoint Information

Currently in Sailfish OS, `udev` starts after `initrd`, which leaves us not being able to use generic partition names (independent of partition number).

In `initrd` we then have to specify hardcoded `/dev/mmcblkXpY` nodes for `/boot` and `/data` partitions.

After `initrd`, `systemd` needs to mount all other required partitions (such as `/system`, `/firmware`, `/persist`, `/config`, `...`) for the HAL layer to work. The required partitions are read from `*.fstab` and `init*.rc` files, disabled there, and respective `.mount` units created – all done by `$ANDROID_ROOT/rpm (droid-hal-device)`.

Unfortunately, `systemd` cannot recognise named partition paths in `.mount` units, because of the same late start of `udev`, even though one can see already created nodes under `/dev/block/platform/*/by-name/` or `/dev/block/platform/*/*/by-name`.

To work around this, we need to create a map between partition names and numbers in `hybris/hybris-boot/fixup-mountpoints` for each device, for all partitions – in this way we are sure to cover them all, because if done manually by looking through fstab/rc files, some might get unnoticed.

To get that mapping, you should boot to CM and execute `adb shell` on your host and this: `ls -l /dev/block/platform/*/by-name/` on your device. In case that yielded no results try `ls -l /dev/block/platform/*/*/by-name/` in some cases you could also try `ls -l /dev/block/bootdevice/by-name/`.

Once you've patched `fixup-mountpoints`, take care if you ever have to run `repo sync --fetch-submodules` again because it will reset your changes, unless the file `.repo/local_manifests/$DEVICE.xml` is pointing `hybris-boot` to your fork with the needed fixup-mountpoints changes.

Then when you get to boot to the Sailfish OS UI, please don't forget to upstream your `fixup-mountpoints` patch.

5.4 Building Relevant Bits of CyanogenMod

In the Android build tree, run the following in a `bash` shell (if you are using e.g. `zsh`, you need to run these commands in a `bash` shell, as the Android build scripts are assuming you are running `bash`).

You'll probably need to iterate this a few times to spot missing repositories, tools, configuration files and others:

```
HABUILD_SDK $

source build/envsetup.sh
export USE_CCACHE=1

breakfast $DEVICE

make -jXX hybris-hal
```

The relevant output bits will be in `out/target/product/$DEVICE/`, in particular:

- `hybris-boot.img`: Kernel and initrd
- `hybris-recovery.img`: Recovery boot image
- `system/` and `root/`: HAL system libraries and binaries

The expected disk usage by the source and binaries after `make hybris-hal` is **19 GB** (as of 2015-09-09, hybris-11.0 branch).

5.4.1 Kernel config

Once the kernel has built you can check the kernel config. You can use the Mer kernel config checker:

```
HABUILD_SDK $

cd $ANDROID_ROOT

hybris/mer-kernel-check/mer_verify_kernel_config \
    ./out/target/product/$DEVICE/obj/KERNEL_OBJ/.config
```

Apply listed modifications to the defconfig file that CM is using. Which one? It's different for every device, most likely first:

- Check the value of `TARGET_KERNEL_CONFIG` under $ANDROID_ROOT/device/$VENDOR/*/BoardConfig*.mk
- Examine the output of *make bootimage* for which defconfig is taken when you're building kernel, e.g.: `make -C kernel/lge/hammerhead ... cyanogenmod_hammerhead_defconfig`
- Check CM kernel's commit history of the `arch/arm/configs` folder, look for defconfig

If you are in a rush, get rid only of `ERROR` cases first, but don't forget to come back to the `WARNING` ones too. After you'll have applied the needed changes, re-run `make hybris-boot` and re-verify. Lather, rinse, repeat :) Run also `make hybris-recovery` in the end when no more errors.

Contribute your mods back

Fork the kernel repo to your GitHub home (indicated by `myname` in this doc).

For Nexus 5 with CM 11.0 as base, the next action would be (rename where appropriate to match your device/branch):

```
HABUILD_SDK $

cd kernel/lge/hammerhead
git checkout -b hybris-11.0

DEFCONFIG=arch/arm/configs/cyanogenmod_hammerhead_defconfig

git add $DEFCONFIG

git commit -m "Hybris-friendly defconfig"
git remote add myname https://github.com/myname/android_kernel_lge_hammerhead
git push myname hybris-11.0
```

Create PR to the forked kernel repo under github/mer-hybris. Ask a mer-hybris admin to create one, if it isn't there.

Adjust your `.repo/local_manifests/$DEVICE.xml` by replacing the line

```
<project path="kernel/lge/hammerhead"
  name="CyanogenMod/android_kernel_lge_hammerhead"
  revision="stable/cm-11.0-XNG3C" />
```

with

```
<project path="kernel/lge/hammerhead"
  name="myname/android_kernel_lge_hammerhead"
  revision="hybris-11.0" />
```

5.5 Common Pitfalls

- If `repo sync --fetch-submodules` fails with a message like *fatal: duplicate path device/samsung/smdk4412-common in /home/nemo/android/.repo/manifest.xml*, remove the local manifest with `rm .repo/local_manifests/roomservice.xml`
- If you notice `git clone` commands starting to write out *"Forbidden . . . "* on github repos, you might have hit API rate limit. To solve this, put your github credentials into `~/.netrc`. More info can be found following this link: Perm.auth. with Git repositories
- *error: Cannot fetch . . . (GitError: –force-sync not enabled; cannot overwrite a local work tree.*, usually happens if `repo sync --fetch-submodules` gets interrupted. It is a bug of the repo tool. Ensure all your changes have been safely stowed (check with `repo status`), and then workaround by:

```
HABUILD_SDK $

repo sync --force-sync

repo sync --fetch-submodules
```

- In some cases (with parallel builds), the build can fail, in this case, use `make -j1 hybris-hal` to retry with a non-parallel build and see the error message without output from parallel jobs. The build usually ends with the following output:

```
HABUILD_SDK $

...
Install: .../out/target/product/$DEVICE/hybris-recovery.img
...
Install: .../out/target/product/$DEVICE/hybris-boot.img
```


CHAPTER

SIX

SETTING UP SCRATCHBOX2 TARGET

It is necessary to setup a Scratchbox2 target to use for packaging your hardware adaptation packages in the next section. Download and create your Scratchbox2 target following this wiki:

Important: Please use the 3.0.0 target or newer (as well as the Platform SDK Chroot)

Warning: To ensure consistency with HADK build scripts, name your tooling `SailfishOS-3.0.0` (or whichever release you are building for) instead of wiki's suggested `SailfishOS-latest`, and your target as `$VENDOR-$DEVICE-$PORT_ARCH` (instead of `SailfishOS-latest-armv7hl`). Ignore the i486 target.

https://sailfishos.org/wiki/Platform_SDK_Target_Installation

To verify the correct installation of the Scratchbox2 target, cross-compile a simple "Hello, World!" C application with `sb2`:

```
PLATFORM_SDK $

cd $HOME
cat > main.c << EOF
#include <stdlib.h>
#include <stdio.h>

int main(void) {
  printf("Hello, world!\n");
  return EXIT_SUCCESS;
}
EOF

sb2 -t $VENDOR-$DEVICE-$PORT_ARCH gcc main.c -o test
```

If the compilation was successful you can test the executable by running the following command (this will run the executable using `qemu` as emulation layer, which is part of the `sb2` setup):

```
sb2 -t $VENDOR-$DEVICE-$PORT_ARCH ./test
```

The above command should output "Hello, world!" on the console, this proves that the target can compile binaries and execute them for your architecture.

CHAPTER

SEVEN

PACKAGING DROID HAL

In this chapter, we will package the build results of *Building the Android HAL* as RPM packages and create a local RPM repository. From there, the RPM packages can be added to a local target and used to build libhybris and the QPA plugin. They can also be used to build the rootfs.

7.1 Creating Repositories for a New Device

If the folders `rpm, hybris/droid-configs, hybris-droid-hal-version-$DEVICE` do not exist yet, create them as follows (example is for Nexus 5, adjust as appropriate and push to your GitHub home):

```
PLATFORM_SDK $

cd $ANDROID_ROOT
mkdir rpm
cd rpm
git init
git submodule add https://github.com/mer-hybris/droid-hal-device dhd
# Rename 'hammerhead' and other values as appropriate
sed -e "s/@DEVICE@/hammerhead/" \
    -e "s/@VENDOR@/lge/" \
    -e "s/@DEVICE_PRETTY@/Nexus 5/" \
    -e "s/@VENDOR_PRETTY@/LG/" \
    dhd/droid-hal-@DEVICE@.spec.template > droid-hal-hammerhead.spec
# Please review droid-hal-hammerhead.spec before committing!
git add .
git commit -m "[dhd] Initial content"
# Create this repository under your GitHub home
git remote add myname https://github.com/myname/droid-hal-hammerhead.git
git push myname master
cd -

mkdir -p hybris/droid-configs
cd hybris/droid-configs
git init
git submodule add https://github.com/mer-hybris/droid-hal-configs \
    droid-configs-device
mkdir rpm
sed -e "s/@DEVICE@/hammerhead/" \
    -e "s/@VENDOR@/lge/" \
    -e "s/@DEVICE_PRETTY@/Nexus 5/" \
    -e "s/@VENDOR_PRETTY@/LG/" \
    droid-configs-device/droid-config-@DEVICE@.spec.template > \
    rpm/droid-config-hammerhead.spec
# Please review rpm/droid-config-hammerhead.spec before committing!
git add .
git commit -m "[dcd] Initial content"
# Create this repository under your GitHub home
```

(continues on next page)

(continued from previous page)

```
git remote add myname https://github.com/myname/droid-config-hammerhead.git
git push myname master
cd -

rpm/dhd/helpers/add_new_device.sh
# On Nexus 5 the output of the last command is:
# Creating the following nodes:
# sparse/
# patterns/
# patterns/jolla-configuration-hammerhead.yaml
# patterns/jolla-hw-adaptation-hammerhead.yaml
cd hybris/droid-configs
COMPOSITOR_CFGS=sparse/var/lib/environment/compositor
mkdir -p $COMPOSITOR_CFGS
cat <<EOF >$COMPOSITOR_CFGS/droid-hal-device.conf
# Config for $VENDOR/$DEVICE
EGL_PLATFORM=hwcomposer
QT_QPA_PLATFORM=hwcomposer
# Determine which node is your touchscreen by checking /dev/input/event*. WRITE␣
↪ALL IN ONE LINE(:
LIPSTICK_OPTIONS=-plugin evdevtouch:/dev/input/event0 -plugin␣
↪evdevkeyboard:keymap=/usr/share/qt5/keymaps/droid.qmap
EOF
git add .
git commit -m "[dcd] Patterns and compositor config"
git push myname master
cd -

mkdir -p hybris/droid-hal-version-hammerhead
cd hybris/droid-hal-version-hammerhead
git init
git submodule add https://github.com/mer-hybris/droid-hal-version
mkdir rpm
sed -e "s/@DEVICE@/hammerhead/" \
    -e "s/@VENDOR@/lge/" \
    -e "s/@DEVICE_PRETTY@/Nexus 5/" \
    -e "s/@VENDOR_PRETTY@/LG/" \
    droid-hal-version/droid-hal-version-@DEVICE@.spec.template > \
    rpm/droid-hal-version-hammerhead.spec
# Please review rpm/droid-hal-version-hammerhead.spec before committing!
git add .
git commit -m "[dvd] Initial content"
# Create this repository under your GitHub home
git remote add myname \
    https://github.com/myname/droid-hal-version-hammerhead.git
git push myname master
```

Now to complete you local manifest, this is how it would be done for Nexus 5. Do it for your device by renaming accordingly:

```
# add the next 3 entries into .repo/local_manifests/hammerhead.xml

<project path="rpm/"
         name="myname/droid-hal-hammerhead" revision="master" />
<project path="hybris/droid-configs"
         name="myname/droid-config-hammerhead" revision="master" />
<project path="hybris/droid-hal-version-hammerhead"
         name="myname/droid-hal-version-hammerhead" revision="master" />
```

Once all these 3 repositories get upstreamed under https://github.com/mer-hybris create PR into an appropriate branch of the file `.repo/local_manifests/hammerhead.xml` to the https://github.com/mer-hybris/local_manifests repository.

7.2 Packaging droid-hal-device

The $ANDROID_ROOT/rpm/ dir contains the needed .spec file to make a set of RPM packages that form the core Droid hardware adaptation part of the hardware adaptation. It also builds a development package (ends with -devel) that contains libraries and headers, which are used when building middleware components later on.

7.2.1 Building the droid-hal-device packages

The next step has to be carried out in the Platform SDK chroot:

```
PLATFORM_SDK $

cd $ANDROID_ROOT

rpm/dhd/helpers/build_packages.sh
```

This should compile all the needed packages, patterns, middleware and put them under local repository. If anything needs modified, just re-run this script.

7.2.2 Troubleshoot errors from build_packages.sh

- **Installed (but unpackaged) file(s) found**: Add those files to straggler section in your rpm/droid-hal-$DEVICE.spec before the %include ... line, for example:

```
%define straggler_files \
/init.mmi.boot.sh\
/init.mmi.touch.sh\
/init.qcom.ssr.sh\
/selinux_version\
/service_contexts\
%{nil}
```

- Then add - droid-hal-hammerhead-detritus to droid-configs/patterns/jolla-hw-adaptation-hammerhead.yaml (substitute as appropriate for your device)
- Lastly, re-run build_packages.sh

CHAPTER

EIGHT

CREATING THE SAILFISH OS ROOT FILESYSTEM

8.1 Additional Packages for Hardware Adaptation

Some additional packages are used to allow access to device features. These middleware packages are usually built against droid-headers / libhybris, and therefore need to be built separately for each target device.

See *Middleware* for a list of all middleware components (not all middleware components are used for all device adaptations). Most of them will have already been built by the `build_packages.sh` script, but if you need a extra one, clone its repository from Github and rebuild via `rpm/dhd/helpers/build_packages.sh --mw=GIT_URL`.

Via the flexible system of patterns, you will be able to select only working/needed functions for your device.

8.2 Allowed Content in Your Sailfish OS Image

The default set of packages results in a minimal and functional root filesystem.

It is forbidden to add proprietary/commercial packages to your image, because royalty fees need to be paid or licence constraints not allowing to redistribute them. Examples:

- jolla-xt9 (dictionary suggestions while typing)
- sailfish-eas (Microsoft Exchange support)
- aliendalvik (Android runtime support)
- sailfish-maps
- Any non-free audio/video codecs, etc.

8.3 Creating and Configuring the Kickstart File

The kickstart file is already generated by the `build_packages.sh` script, during droid-configs build, using `ssuks`, which is part of the SSU utility:

```
PLATFORM_SDK $

cd $ANDROID_ROOT

HA_REPO="repo --name=adaptation-community-common-$DEVICE-@RELEASE@"
HA_DEV="repo --name=adaptation-community-$DEVICE-@RELEASE@"
KS="Jolla-@RELEASE@-$DEVICE-@ARCH@.ks"
sed \
 "/$HA_REPO/i$HA_DEV --baseurl=file:\/\/$ANDROID_ROOT\/droid-local-repo\/$DEVICE" \
 $ANDROID_ROOT/hybris/droid-configs/installroot/usr/share/kickstarts/$KS \
 > $KS
```

8.4 Patterns

The selection of packages for each hardware adaptation has to be put into a pattern file, so that creating the image as well as any system updates in the future can pull in and upgrade all packages related to the hardware adaptation.

8.4.1 Modifying a pattern

To make an extra modification to a pattern, edit its respective file under `hybris/droid-configs/patterns/`. Take care and always use `git status/stash` commands. Once happy, commit to your GitHub home and eventually PR upstream.

For patterns to take effect on the image, run the following:

```
PLATFORM_SDK $

cd $ANDROID_ROOT
rpm/dhd/helpers/build_packages.sh --configs
```

NB: it will fail with a non-critical `Exception AttributeError: "'NoneType...` error.

8.5 Building the Image with MIC

In the script below choose a Sailfish OS version you want to build.

Important: Avoid building older releases unless you know what you're doing - we do not guarantee backwards compatibility for old Sailfish OS versions! E.g., expect patterns to break as new HA packages get introduced etc.

Ensure you pick the same release as your target was in *Setting up Scratchbox2 Target*. E.g., if target's `ssu lr` versions begin with `2.1.1.`, build Sailfish OS update 2.1.1.26 (check for the latest, non "early access" Sailfish OS version)

Build a rootfs using RPM repositories and a kickstart file (NB: all errors are non-critical as long as you end up with a generated .zip image):

```
PLATFORM_SDK $

# Set the version of your choosing, latest is strongly preferred
# (check with "Sailfish OS version" link above)
RELEASE=2.1.1.26
# EXTRA_NAME adds your custom tag. It doesn't support '.' dots in it!
EXTRA_NAME=-my1
# Always regenerate patterns as they usually get reset during build process
# NB: the next command will output a non-error, safe to ignore it:
# Exception AttributeError: "'NoneType' object has no attribute 'px_proxy_fa..
hybris/droid-configs/droid-configs-device/helpers/process_patterns.sh
sudo mic create fs --arch=$PORT_ARCH \
    --tokenmap=ARCH:$PORT_ARCH,RELEASE:$RELEASE,EXTRA_NAME:$EXTRA_NAME \
    --record-pkgs=name,url \
    --outdir=sfe-$DEVICE-$RELEASE$EXTRA_NAME \
    --pack-to=sfe-$DEVICE-$RELEASE$EXTRA_NAME.tar.bz2 \
    $ANDROID_ROOT/Jolla-@RELEASE@-$DEVICE-@ARCH@.ks
```

Once obtained the `.zip` file, sideload via your device's recovery mode, or examine other particular ways of deploying to your device.

Jolla Store functionality can be enabled only if your device identifies itself uniquely - either via IMEI or (for non-cellular devices) WLAN/BT MAC address. Consult us on #sailfishos-porters IRC channel on Freenode.net about details.

If creation fails due to absence of a package required by pattern, note down the package name and proceed to *Dealing with a Missing Package*.

A more obscure error might look like this:

```
Warning: repo problem: pattern:jolla-configuration-$DEVICE-(version).noarch
  requires jolla-hw-adaptation-$DEVICE,
  but this requirement cannot be provided, uninstallable providers:
  pattern:jolla-hw-adaptation-$DEVICE-(version).noarch[$DEVICE]
```

This means a package dependency cannot be satisfied down the hierarchy of patterns. A quick in-place solution (NB: expand @DEVICE@ occurrences manually):

- Substitute the line `@Jolla Configuration @DEVICE@` with `@jolla-hw-adaptation-@DEVICE@` in your .ks
- Update patterns (*Modifying a pattern*)
- Try creating the image again (*Building the Image with MIC*)
- Repeat the steps above substituting respective pattern to walk down the patterns hierarchy – you'll eventually discover the offending package
- If that package is provided by e.g. droid-hal-device (like `droid-hal-mako-pulseaudio-settings`), it means that some of its dependencies are not present:
- Edit .ks file by having `%packages` section consisting only of single `droid-hal-mako-pulseaudio-settings` (note there is no @ at the beginning of the line, since it's a package, not a pattern) – another `mic` run error will show that the offending package is actually `pulseaudio-modules-droid`

Important: When found and fixed culprit in next sections, restore your .ks `%packages` section to `@Jolla Configuration @DEVICE@`! Then try creating the image again (*Building the Image with MIC*)

Now you're ready to proceed to the *Dealing with a Missing Package* section.

8.5.1 Dealing with a Missing Package

If that package is critical (e.g. `libhybris`, `qt5-qpa-hwcomposer-plugin` etc.), build and add it to the local repo as explained in extra-mw. Afterwards perform:

- *Modifying a pattern*
- *Building the Image with MIC*

Otherwise if a package is not critical, and you accept to have less functionality (or even unbootable) image, you can temporarily comment it out from patterns in `hybris/droid-configs/patterns` and orderly perform:

- *Modifying a pattern*
- *Building the Image with MIC*

Alternatively (or if you can't find it among patterns) provide a line beginning with dash (e.g. `-jolla-camera`) indicating explicit removal of package, to your .ks `%packages` section (remember that regenerating .ks will overwrite this modification).

8.5.2 Troubleshooting

/dev/null - Permission denied

Most likely the partition your Platform SDK resides in, is mounted with `nodev` option. Remove that option from mount rules.

CHAPTER

NINE

GETTING IN

9.1 Boot and Flashing Process

This varies from device to device. There are a few different boot loaders and flashing mechanisms used for Android devices:

- **fastboot**: Used by most Nexus devices
- **odin**: Used by most Samsung devices

For flashing fastboot-based devices, use `fastboot` (available in the Platform SDK), for odin-based devices, use Heimdall.

9.2 Operating Blind on an Existing Device

Long story short, you will have to assume that you cannot:

- See any framebuffer console
- See any error messages of any kind during bootup
- Get any information relayed from your startup process
- Set any kind of modified kernel command lines

Hence, we have to learn how to operate blind on a device. The good news is that when you have a working kernel, you can combine it with a init ramdisk and that Android's USB gadget is built in to most kernel configurations. It is possible then for the ramdisk to set up working USB networking on most devices and then open up a telnet daemon.

The **hybris-boot** repository contains such an initrd with convenient USB networking, DHCP and telnet server, plus the ability to boot into a Sailfish OS system. The init system in the hybris-boot initrd will attempt to write information via the USB device serial number and model. So `dmesg` on the host could produce:

```
HOST $

dmesg    # sample output:
...
[1094634.238136] usb 2-2: Manufacturer: Mer Boat Loader
[1094634.238143] usb 2-2: SerialNumber: Mer Debug setting up (DONE_SWITCH=no)
...
```

However `dmesg` doesn't report all changes in the USB subsystem and the init script will attempt to update the iSerial field with information so also do:

```
HOST $

lsusb -v | grep iSerial   # sample output:
 iSerial    3 Mer Debug telnet on port 23 on rndis0 192.168.2.15 - also running␣
↪udhcpd
```

(continues on next page)

(continued from previous page)

However, if it says something like:

```
[1094634.238143] usb 2-2: SerialNumber: Mer Debug setting up (DONE_SWITCH=yes)
```

connectivity will be available via `telnet 192.168.2.15 2323` port.

9.3 Logs across reboots

```
DEVICE $

devel-su
# change Storage=volatile --> Storage=automatic in:
vi /etc/systemd/journald.conf
mkdir /var/log/journal
reboot
```

Systemd suppresses journal, and some valuable info might get hidden. To prevent this, set `RateLimitInterval=0`

9.3.1 Bootloops

If device bootloops, there might be several reasons:

- If it immediately reboots (and especially if it later boots to recovery mode), SELinux is enabled, and all ports based on Android 4.4 or newer need to disable it. Add `CONFIG_SECURITY_SELINUX_BOOTPARAM=y` to your kernel defconfig, and `selinux=0` to your kernel command line (usually in `BOARD_KERNEL_CMDLINE` under $ANDROID_ROOT/device/$VENDOR/*/BoardConfig*.mk)
- If it reboots after a minute or so, be quick and telnet into device, then do:

  ln -s /dev/null /etc/systemd/system/ofono.service
- Check if your /system is mounted by systemd (system.mount unit)

9.3.2 Tips

To ease debugging in unstable/halting/logs spamming early ports:

```
DEVICE $

systemctl mask droid-hal-init
systemctl mask user@100000
```

9.3.3 Get connected

Use USB networking to connect to the Internet from your Sailfish OS

Execute on your host as root. Use the interface which your host uses to connect to the Internet. It's wlan0 in this example:

```
HOST $

iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
```


Execute on the device:

```
TARGET $

route add default gw 192.168.2.X # <- host's usb0 IP
echo 'nameserver 208.67.222.222' > /etc/resolv.conf
```

9.4 Splitting and Re-Assembling Boot Images

A **boot.img** file is basically a combination of a Linux kernel and an initramfs as `cpio` archive. The Platform SDK offer the `mkbootimg` to build a boot image from a kernel and cpio archive. To split a boot image, use `split_bootimg` in Platform SDK.

In the CyanogenMod-based Sailfish OS port, a boot image with Sailfish OS- specific scripts will be built automatically. These boot images are then available as **hybris-boot.img** (for booting into Sailfish OS) and **hybris-recovery.img** (for debugging via telnet and test-booting).

CHAPTER

TEN

FLASHING THE ROOTFS IMAGE

In order to be able to use Sailfish OS on the device, the parts that we built and assembled in the previous chapters now need to be flashed to the device. After flashing, Sailfish OS should boot on your device on the next reboot.

10.1 Prerequisites

- Android Recovery flashed to your device
- The stock firmware image (for your version and device)
- The vanilla CM release (for your version and device)
- A Sailfish OS rootfs update .zip, created by `mic`

10.2 Flashing back to Stock Android

It is important that you start with a fresh stock image that matches the Android version of the CyanogenMod release you are going to flash (which in turn is dictated by the Sailfish OS image you are going to flash).

While the CM .zip contains all files in `/system/` (e.g. libraries and libhardware modules), the stock image also contains firmware parts and flashables for partitions that are not included in the CM .zip.

For example, if you are running stock 4.4.2 on a Nexus 4 (mako), and you are going to flash CM 10.1.3 and Sailfish OS to it, you have to first flash the stock 4.2.2 (note that this is 4.2, not 4.4) first, so that the firmware bits are matching the CM version.

If you do not flash the right stock version (and therefore firmware), there might be some issues when booting into Sailfish OS:

- Problems accessing `/sdcard/` in recovery (e.g. `adb push` does not work)
- WLAN, sensors, audio and other hardware not working

If you experience such issues, please make sure you first flash the stock system, ROM, followed by a Recovery image and CyanogenMod, and finally the Sailfish OS update. Please also note that you can't just take the latest stock ROM and/or CyanogenMod ROM - both versions have to match the Sailfish OS version you are going to install, as the Sailfish OS parts are built against a specific version of the HA.

10.3 Flashing using Android Recovery

1. Boot into Android Recovery
2. Upload the CM release: `adb push cm-10.1.3-$DEVICE.zip /sdcard/`
3. Upload Sailfish OS: `adb push sailfishos-$DEVICE-devel-1.2.3.4.zip /sdcard/`
4. In the Recovery on the device:

1. Clear data and cache (factory reset)
2. Install the CM release by picking the CM image
3. Install Sailfish OS by picking the SFOS image
4. Reboot the device

CHAPTER

ELEVEN

MANUAL INSTALLATION AND MAINTENANCE

This assumes you are booted into CyanogenMod on your device, can `adb shell` to it to get a root shell and have your boot image and rootfs tarball ready.

Some of these approaches also work in Android Recovery (there's an `adbd` running), but you obviously won't have network connectivity for downloading updates.

11.1 Extracting the rootfs via adb

Replace `i9305-devel.tar.gz` with the name of your rootfs tarball:

```
PLATFORM_SDK $

adb push i9305-devel.tar.gz /sdcard/
adb shell
su
mkdir -p /data/.stowaways/sailfishos
tar --numeric-owner -xvzf /sdcard/i9305-devel.tar.gz \
    -C /data/.stowaways/sailfishos
```

11.2 Flashing the boot image via adb

The following example is for `i9305`, for other devices the output partition and filename is obviously different:

```
PLATFORM_SDK $

adb push out/target/product/i9305/hybris-boot.img /sdcard/
adb shell
su
dd if=/sdcard/hybris-boot.img of=/dev/block/mmcblk0p8
```

11.3 Interacting with the rootfs via adb from Android

You can interact with the Sailfish OS rootfs and carry out maintenance (editing files, installing packages, etc..) when booted into an Android system. You have to have your rootfs already installed/extracted. You can use Android's WLAN connectivity to connect to the Internet and download updates:

```
PLATFORM_SDK $

adb shell
su
mount -o bind /dev /data/.stowaways/sailfishos/dev
```

(continues on next page)

(continued from previous page)

```
mount -o bind /proc /data/.stowaways/sailfishos/proc
mount -o bind /sys /data/.stowaways/sailfishos/sys
chroot /data/.stowaways/sailfishos/ /bin/su -
echo "nameserver 8.8.8.8" >/etc/resolv.conf
...
```


CHAPTER

TWELVE

MODIFICATIONS AND PATCHES

Running Sailfish OS on top of a Mer Hybris adaptation requires a few modifications to a standard Android/CM system. We maintain forks of some repos with those patches aplied.

12.1 Hybris Modifications to CyanogenMod

Our modifications are tracked by our own Hybris-specific repo manifest file. The below sections outline our modifications to these sources.

12.1.1 Droid System

In order to work with `libhybris`, some parts of the lower levels of Android need to be modified:

- **bionic/**
  - Pass `errno` from bionic to libhybris (`libdsyscalls.so`)
  - Rename `/dev/log/` to `/dev/alog/`
  - TLS slots need to be re-assigned to not conflict with glibc
  - Support for `HYBRIS_LD_LIBRARY_PATH` in the linker
  - Add `/usr/libexec/droid-hybris/system/lib` to the linker search path
- **external/busybox/: Busybox is used in the normal and recovery boot images. We need** some additional features like `mdev` and `udhcpd`
- **system/core/**
  - Make `cutils` and `logcat` aware of the new log location (`/dev/alog/`)
  - Add `/usr/libexec/droid-hybris/lib-dev-alog/` to the `LD_LIBRARY_PATH`
  - Force SELinux OFF since hybris does not utilise the relevant Android parts, and leaving SELinux support ON would then cause device to reboot to recovery
  - Remove various `init` and `init.rc` settings and operations that are handled by `systemd` and/or Hybris on a Sailfish OS system
- **frameworks/base/: Only build `servicemanager`, `bootanimation` and `androidfw`** to make the minimal Droid HAL build smaller (no Java content)
- **libcore/**: Don't include `JavaLibrary.mk`, as Java won't be available

All these modifications have already been done in the **mer-hybris** GitHub organisation of forks from the original CyanogenMod sources. If its `android` manifest is used, these changes will be included automatically.

In addition to these generic modifications, for some devices and SoCs we also maintain a set of patches on top of CyanogenMod to fix issues with drivers that only happen in Sailfish OS, for example:

- **hardware/samsung/**: SEC hwcomposer: Avoid segfault if `registerProcs` was never called

12.1.2 Kernel

For the Kernel, some configuration options must be enabled to support `systemd` features, and some configuration options must be disabled, because they conflict or block certain features of Sailfish OS.

- **Required Configuration Options:** See `$ANDROID_ROOT/hybris/hybris-boot/init-script` function `check_kernel_config()` for a list of required kernel options
- **Conflicting Configuration Options:** **`CONFIG_ANDROID_PARANOID_NETWORK`:** This would make all network connections fail if the user is not in the group with ID 3003.

As an alternative to checking the kernel options in the initramfs, the script `$ANDROID_ROOT/hybris/mer-kernel-check` can also be used to verify if all required configuration options have been enabled.

12.2 Configuring and Compiling the Kernel

For supported devices, the kernel is built as part of `mka hybris-hal` with the right configuration.

For new devices, you have to make sure to get the right kernel configuration included in the repository. For this, clone the kernel repository for the device into **mer-hybris** and configure the kernel using `hybris/mer-kernel-check`.

CHAPTER THIRTEEN

DETAILED SUBSYSTEM ADAPTATION GUIDES

Mer / Sailfish OS uses some kernel interfaces directly, bypassing the android HAL. Mainly this is used in places where the kernel API is stable enough and also used by Android. The other reasons for using kernel APIs directly include better features offered by standard kernel frameworks, differing middleware between Mer / Sailfish OS linux and Android, and lastly special features of Sailfish OS.

13.1 Vibration / force feedback

The default vibra framework that is used in full featured productized Sailfish OS devices is the force feedback API in kernel input framework. The kernel drivers should either use the ffmemless framework OR provide FF_PERIODIC and FF_RUMBLE support via as a normal input driver. In this chapter we go through the ff-memless aproach of adapting your kernel for Mer/Sailfish OS

This is a different method than what is used in community Sailfish OS ports, which utilize the android vibrator / timed-output API. The android vibrator plugins in Mer/Sailfish OS middleware have very reduced feature set, and are not recommended for commercial products.

In order to utilize the standard input framework force feedback features of Sailfish OS, the android timed output vibrator kernel driver needs to be converted to a ffmemless driver. The main tasks for this are:

- Enable CONFIG_INPUT_FF_MEMLESS kernel config option
- Disable CONFIG_ANDROID_TIMED_OUTPUT kernel config option
- Change maximum amount of ffmemless effects to **64** by patching ff-memless.c:
- http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/input/ff-memless.c#n41

```
diff --git a/drivers/input/ff-memless.c b/drivers/input/ff-memless.c
index 117a59a..fa53611 100644
--- a/drivers/input/ff-memless.c
+++ b/drivers/input/ff-memless.c
@@ -39,7 +39,7 @@ MODULE_AUTHOR("Anssi Hannula <anssi.hannula@gmail.com>");
 MODULE_DESCRIPTION("Force feedback support for memoryless devices");

 /* Number of effects handled with memoryless devices */
-#define FF_MEMLESS_EFFECTS     16
+#define FF_MEMLESS_EFFECTS     64

 /* Envelope update interval in ms */
 #define FF_ENVELOPE_INTERVAL   50
```

- Optionally you can decrease ff-memless control interval so that fade and attack envelopes can be used in short haptic effects as well:

```
diff --git a/drivers/input/ff-memless.c b/drivers/input/ff-memless.c
index 89d3a3d..33eee2e 100644
--- a/drivers/input/ff-memless.c
```

(continues on next page)

(continued from previous page)

```
+++ b/drivers/input/ff-memless.c
@@ -41,7 +41,7 @@ MODULE_DESCRIPTION("Force feedback support for memoryless devi
 #define FF_MEMLESS_EFFECTS     64

 /* Envelope update interval in ms */
-static int ff_envelope_interval = 50;
+static int ff_envelope_interval = 10;
 module_param(ff_envelope_interval, int, S_IWUSR | S_IRUGO);

 #define FF_EFFECT_STARTED      0
```

- If your platform happens to already support a ffmemless based vibra driver, just enable it and fix any issues that you see. Otherwise go through the rest of the points below.
- Convert the android timed output vibra driver to support to ffmemless
  - add "#include <linux/input.h>"
  - Create a ffmemless play function.
  - Examples of ffmemless play functions / ffmemless drivers:
  - http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/input/misc/arizona-haptics.c#n110
  - http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/input/misc/max8997_haptic.c#n231
  - http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/input/misc/pm8xxx-vibrator.c#n130
  - At probe, create a ffmemless device with **input_ff_create_memless**
  - http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/include/linux/input.h#n531
  - And register the resulting device with input_device_register.
  - Remember to clean up the input device structure at driver exit
  - The example ffmemless drivers above can be used for reference

The userspace configuration haptic feedback and effects is handled with ngfd configuration files, see more details in

- *Configuring haptics*

13.2 GStreamer v1.0

Sailfish OS 2.0 introduces GStreamer v1.0 with hardware-accelerated video and audio encoding and decoding in Camera, Gallery and Browser, and deprecates GStreamer v0.10.

```
HABUILD_SDK $

cd $ANDROID_ROOT
source build/envsetup.sh
breakfast $DEVICE
make -jXX $(external/droidmedia/detect_build_targets.sh $PORT_ARCH
↪$(gettargetarch))
```

Note: If during intense development you need to rebuild droidmedia multiple times, you can quicken by executing `gettargetarch > lunch_arch` once, then running `make` without the `$(gettargetarch)` param.


```
PLATFORM_SDK $

cd $ANDROID_ROOT
DROIDMEDIA_VERSION=$(git --git-dir external/droidmedia/.git describe --tags | sed \
  -r "s/\-/\+/g")
rpm/dhd/helpers/pack_source_droidmedia-localbuild.sh $DROIDMEDIA_VERSION
mkdir -p hybris/mw/droidmedia-localbuild/rpm
cp rpm/dhd/helpers/droidmedia-localbuild.spec \
  hybris/mw/droidmedia-localbuild/rpm/droidmedia.spec
sed -ie "s/0.0.0/$DROIDMEDIA_VERSION/" \
  hybris/mw/droidmedia-localbuild/rpm/droidmedia.spec
mv hybris/mw/droidmedia-$DROIDMEDIA_VERSION.tgz hybris/mw/droidmedia-localbuild
rpm/dhd/helpers/build_packages.sh --build=hybris/mw/droidmedia-localbuild
```

Build relevant parts:

```
PLATFORM_SDK $

cd $ANDROID_ROOT
rpm/dhd/helpers/build_packages.sh --droid-hal --mw=https://github.com/sailfishos/
↪gst-droid.git
```

Add the GStreamer-droid bridge to patterns in `$ANDROID_ROOT/hybris/droid-configs/`:

```
diff --git a/patterns/jolla-hw-adaptation-$DEVICE.yaml b/patterns/jolla-hw-
↪adaptation-$DEVICE.yaml
 - nemo-gstreamer1.0-interfaces
+- gstreamer1.0-droid
+
 # This is needed for notification LEDs
 - mce-plugin-libhybris
```

Rebuild configs and patterns:

```
PLATFORM_SDK $

cd $ANDROID_ROOT
rpm/dhd/helpers/build_packages.sh --configs
```

You are now ready to rebuild the image which will have GStreamer v1.0 support, refer to *Creating the Sailfish OS Root Filesystem*. Alternatively you can complete productising other HW areas as described in this chapter.

13.3 Camera

Ensure you have built the *GStreamer v1.0* part in the previous section.

To reduce amount of patches done to the Android BSP (that's the ultimate aim in general), we will always build audioflingerglue middleware, this will also fix phonecalls audio on many adaptations.

```
HABUILD_SDK $

cd $ANDROID_ROOT
source build/envsetup.sh
breakfast $DEVICE
make -jXX $(external/audioflingerglue/detect_build_targets.sh $PORT_ARCH
↪$(gettargetarch))
```

Note: If during intense development you need to rebuild audioflingerglue multiple times, you can quicken by executing `gettargetarch > lunch_arch` once, then running `make` without the `$(gettargetarch)` param.

```
PLATFORM_SDK $

cd $ANDROID_ROOT
rpm/dhd/helpers/pack_source_audioflingerglue-localbuild.sh
mkdir -p hybris/mw/audioflingerglue-localbuild/rpm
cp rpm/dhd/helpers/audioflingerglue-localbuild.spec \
  hybris/mw/audioflingerglue-localbuild/rpm/audioflingerglue.spec
mv hybris/mw/audioflingerglue-0.0.1.tgz hybris/mw/audioflingerglue-localbuild
rpm/dhd/helpers/build_packages.sh --build=hybris/mw/audioflingerglue-localbuild
rpm/dhd/helpers/build_packages.sh --droid-hal \
  --mw=https://github.com/mer-hybris/pulseaudio-modules-droid-glue.git
```

Add the pulseaudio-modules glue package to patterns in `$ANDROID_ROOT/hybris/droid-configs/`:

```
diff --git a/patterns/jolla-hw-adaptation-$DEVICE.yaml b/patterns/jolla-hw-
↪adaptation-$DEVICE.yaml
 - pulseaudio-modules-droid
+- pulseaudio-modules-droid-glue
```

Rebuild configs and patterns:

```
PLATFORM_SDK $

cd $ANDROID_ROOT
rpm/dhd/helpers/build_packages.sh --configs
```

Now you can test the Camera app, it should already work, however with default low settings and reduced feature set (e.g. no flash or focus mode selection).

To improve those, install `gstreamer1.0-droid-tools` on device (RPM is available under `$ANDROID_ROOT/droid-local-repo/$DEVICE/gst-droid/`) and launch:

```
DEVICE $

devel-su # Set your password in Settings | Developer mode
mk-cam-conf 0 /etc/gst-droid/gstdroidcamsrc-0.conf
mk-cam-conf 1 /etc/gst-droid/gstdroidcamsrc-1.conf
```

This creates configs for each, front and back cameras. Transfer them over and place under `$ANDROID_ROOT/hybris/droid-configs/sparse/etc/gst-droid` for persistency (don't forget to git commit+push somewhere safe! :)

Next you'll need to generate the resolutions file. Build the following repo:

```
PLATFORM_SDK $

cd $ANDROID_ROOT
rpm/dhd/helpers/build_packages.sh --mw=droid-camres
```

Install the RPM from `$ANDROID_ROOT/droid-local-repo/$DEVICE/droid-camres/` onto your device and execute:

```
DEVICE $

droid-camres -w

# It creates a failsafe jolla-camera-hw.txt, manual perfecting is encouraged

devel-su # Set your password in Settings | Developer mode
```

(continues on next page)

(continued from previous page)

```
mv jolla-camera-hw.txt /etc/dconf/db/vendor.d/
dconf update
```

Go to Settings | Apps | Camera and ensure valid ratio and megapixel entries appear in both cameras. Reloading Camera app should effectuate the changes.

You can further fix/improve the contents of `jolla-camera-hw.txt` by looking more closely at the output of `droid-camres`. Sometimes it chooses an aspect ratio which provides sub-optimal resolution, e.g. it prefers 4:3 for the front facing camera, yet sensor only supports 1280x960, however switching to 16:9 would give a far superior 1920x1080 resolution.

This command will list all available parameters for a specific camera from the underlying HAL, which will help with tweaking values such as ISO speed, focus and flash:

```
GST_DEBUG=6 mk-cam-conf 0 /dev/null 2>&1 | grep params_parse | sed -e 's/.*param\s/
↪/' | sort -u
```

If you find some parameters (such as ISO speed or other 3A settings) are missing, then it's possible that your camera device is designed to use an older version of the Camera HAL than the default. You can try forcing a HAL v1 connection by adding `FORCE_HAL:=1` to `env.mk` in droidmedia.

You are encouraged to set all viewfinder resolutions to match that of your device's framebuffer. Do check for regressions via `devel-su dconf update` and reloading Camera app as you go.

Preserve `/etc/dconf/db/vendor.d/jolla-camera-hw.txt` under version control just like you did with `gstdroidcamsrc-*.conf` above.

If your device supports flash torch during video recording change `flashValues=[2]` under `[apps/jolla-camera/primary/video]` to `flashValues=[2, 32]`.

Lastly, check other variants of `/etc/dconf/db/vendor.d/jolla-camera-hw.txt` throughout the range of existing Sailfish OS devices, or consult our developers how to obtain e.g. more valid ISO values, focus distance, add other MegaPixel values etc.

Ultimately you are the most welcome to improve the `droid-camres` tool itself by contributing upstream!

13.4 Cellular modem

- Ensure Android's RIL running `ps ax | grep rild` (expect one or two `/system/bin/rild`)
- If RIL is not running, check why it is not launched from /init*.rc scripts
- If it's launched, check where it fails with `/usr/libexec/droid-hybris/system/bin/logcat -b radio`
- Errors in RIL might look like this:

  RIL[0][main] qcril_qmi_modem_power_process_bootup: ESOC node is not available

After online search this suggests firmware loading issues on Motorola Moto G. Compare with a healthy radio logcat after booting back into CM, not all lines starting with `E/RIL...` will point to a root cause!

- If it's firmware loading problem, trace all needed daemons in CM and their loading order as well as all mounted firmware, modem, and baseband partitions.
- Once RIL is happy, then ofono can be launched. Unmask it if it was previously masked due to causing reboots in *Bootloops*.
- If you still get no signal indicator in UI, remove SIM PIN and retry
- Also install `ofono-tests` package and run `/usr/lib/ofono/test/list-modems`
- Try to recompile latest ofono master branch from https://git.merproject.org/mer-core/ofono

- If everything else fails, then stop and strace a failing daemon (either RIL or ofono) from command line manually

13.4.1 Phone calls don't work (but SMS and mobile data works)

`audioflingerglue` middleware is required, which is now always built together with the *Camera* adaptation.

13.5 Bluetooth

For bluetooth Sailfish OS uses BlueZ stack from linux.

TODO: bluetooth adaptation guide.

TODO: add detail about audio routing.

13.6 WLAN

Typically WLAN drivers are external kernel modules in android adaptations. To set up WLAN for such devices, a systemd service file needs to be created that loads the kernel module at boot. In addition to this you need to check that firmware files and possible HW tuning files are installed in correct locations on the filesystem.

Mer / Sailfish OS WLAN adaptation assumes the driver is compatible with WPA supplicant. This means the WLAN device driver has to support cfg80211 interface. In some cases connman (the higher level connection manager in Mer/Sailfish) accesses directly the WLAN driver bypassing wpa_supplicant.

The version of currently used wpa_supplicant can be checked from here:

> https://git.merproject.org/mer-core/wpa_supplicant

The version of used connman can be checked from here:

> https://git.merproject.org/mer-core/connman

13.6.1 Special quirks: WLAN hotspot

On some android WLAN drivers, the whole connectivity stack needs to be reset after WLAN hotspot use. For that purpose there is reset service in dsme, please see details how to set that up for your adaptation project in here:

> https://git.merproject.org/mer-core/dsme/commit/c377c349079b470db38ba6394121b6d899004963

13.7 NFC

Currently there is no NFC middleware in Sailfish OS. Android HAL API support should be enough for future compatibility.

13.8 GPS

Ensure the `test_gps` command gets a fix after a while.

The necessary middleware is already built for you, just add `geoclue-provider-hybris` package into your patterns.

13.9 Audio

For audio, Mer / Sailfish OS uses PulseAudio as the main mixer. For audio routing ohmd is used.

TODO: Add info about audio routing configuration TODO: Add more info in general.

13.10 Sensors

Sailfish OS sensor support is based upon Sensor Framework at: https://git.merproject.org/mer-core/sensorfw

Hybris based systems can use the hybris sensor adaptor plugins, which uses existing android libhardware sensor adaptations to read sensor data and control.

It can also be configured for standard linux sysfs and evdev sensor interfaces.

It should be configured at /etc/sensorfw/primaryuse.conf, which links to a device specific conf file. Historically named sensord-<BOARDNAME>.conf. You can also use any conf file by specifying it on the commandline.

For hybris based platforms, this will be sensord-hybris.conf, and most likely will not have to be modified. A copy of this file is already among default configs: https://git.merproject.org/mer-core/sensorfw/blob/master/config/sensord-hybris.conf If you do make modifications to it, add the file under `$ANDROID_ROOT/hybris/droid-configs/sparse/etc/sensorfw/primaryuse.conf`

There are already a few device specific conf files to look at if the device needs more configuration. Example of mixed hybris and evdev configuration https://git.merproject.org/mer-core/sensorfw/blob/master/config/sensord-tbj.conf

Generally, if sensors are working on the android/hybris side, they will work in sensorfw and up to the Sailfish UI. libhybris comes with /usr/bin/test_sensors which can list those Android sensors found.

Above Sensor Framework is QtSensors, which requires a configuration file at /etc/xdg/QtProject/QtSensors.conf which is supplied with the sensorfw backend plugin in QtSensors and a copy of it is already among your default configs.

For Mer based systems, the QtSensors source code is at: https://github.com/mer-qt/qtsensors

Debugging output of sensorfwd can be increased one level during runtime by sending (as root) USR1 signal like so: kill -USR1 *pgrep sensorfwd* or specified on the commandline for startup debugging.

Sending kill -USR2 *pgrep sensorfwd* will output a current status report.

13.11 Power management

Under the hood, Sailfish OS uses the android wake locks. Typically there is no need to change anything in the kernel side (assuming it works fine with android) for the power management to work, as long as all the device drivers are working normally.

The userspace API's for platform applications is exposed via nemo-keepalive package. See more details here:

> https://git.merproject.org/mer-core/nemo-keepalive

13.12 Watchdog

A standard linux kernel watchdog core driver support is expected. The device node should be in /dev/watchdog. It should be configured with following kernel options:

```
CONFIG_WATCHDOG=y
CONFIG_WATCHDOG_CORE=y
CONFIG_WATCHDOG_NOWAYOUT=y
```


- **NOTE 1**: Please note that watchdog driver should disable itself during suspend.
- **NOTE 2**: Normally the watchdog period is programmed automatically, but if your driver does not support programming the period, the default kicking period is 20 seconds.

13.13 Touch

Sailfish OS is compatible with standard kernel multitouch input framework drivers. Protocol A is preferred. The main configuration needed is to symlink the correct event device node to /dev/touchscreen. To do this the best way is to set up a udev rule that checks the devices with evcap script and creates the link once first valid one is found. See more details for evcap here:

> https://github.com/mer-hybris/evcap

The udev rule can be put to file

> /lib/udev/rules.d/61-touchscreen.rules

The reason this is not done by default is that typically driver authors mark bit varying capabilities as supported and there could be multiple touch controllers on a device, so the final rule is best to be written in a device specific configs package.

NOTE: if you still have problems with touch, please check that lipstick environment has correct touch device parameter:

> cat /var/lib/environment/compositor/droid-hal-device.conf

- **LIPSTICK_OPTIONS** should have **"-plugin evdevtouch:/dev/touchscreen"**

13.13.1 Special feature: double tap to wake up

Sailfish OS supports waking up the device from suspend (unblanking the screen) via double tap gesture to the touchscreen. The touchscreen driver should either emulate KEY_POWER press / release or post a EV_MSC/MSC_GESTURE event with value 0x4 when double tap gesture is detected when waking up from suspend.

In order to avoid excess power drain when device is in pocket facing users skin, some sysfs should be exported to allow disabling the touch screen. The feature requires that the device has a working proximity sensor that can wake up the system when it is suspended (to be able to update touch screen state according to need). To configure MCE that handles this see *MCE configuration*

CHAPTER

FOURTEEN

MIDDLEWARE

This chapter contains some background information about the middleware parts that are part of the Hardware Adapation. Using this info, it should be possible to customize and build the middleware parts for a given device.

14.1 MCE libhybris Plugin

TODO

14.2 MCE configuration

/etc/mce/60-doubletap-jolla.ini

Configures the touchscreen kernel driver sysfs that can be used to disable and enable double tap to wake up feature. Example of it's content:

```
# Configuration for doubletap wakeup plugin
[DoubleTap]
# Path to doubletap wakeup control file
ControlPath=/sys/bus/i2c/drivers/touch_synaptics/3-0020/double_tap_enable
# Value to write when enabling doubletap wakeups
EnableValue=1
# Value to write when Disabling doubletap wakeups
DisableValue=0
```

TODO:

/etc/mce/60-mce-cpu-scaling-governor.ini

/etc/mce/60-mce-display-blank-timeout.conf

/etc/mce/60-mce-display-brightness.conf

/etc/mce/60-mce-possible-display-dim-timeouts.conf

/etc/mce/60-memnotify-jolla.conf

14.3 Configuring haptics

Mer/Sailfish OS has 2 kinds of feedback methods:

1. **NGFD** - Non-graphical feedback framework **ffmemless** plugin
2. **QtFeedback** - QtFeedback with direct ffmemless backend

The NGFD plugin is for providing feedback for events and alarms, while QtFeedback is used for minimum latency haptics and for 3rd party applications.

Both of these have their own default .ini configuration files with the default effects for basic use. The default configurations can be overridden with device specific .ini files in your adaptation project's config package. The default config files can be seen in:

- **NGFD**: /usr/share/ngfd/plugins.d/ffmemless.ini
- **QtFeedback**: /usr/lib/qt5/plugins/feedback/ffmemless.ini

The default configuration files can be over-ridden with setting environment variables NGF_FFMEMLESS_SETTINGS (ngfd) and FF_MEMLESS_SETTINGS (qtfeedback), that point to device specifc configuration files.

To set the environment variables add environment config file to your config package that installs to (**NOTE**: Replace "**DEVICE**" with your device's name. E.g. mako, hammerhead, etc.):

/var/lib/environment/nemo/60-DEVICE-vibra.conf

And that file should contain 2 lines:

```
FF_MEMLESS_SETTINGS=/usr/lib/qt5/plugins/feedback/qtfeedback-DEVICE.ini
NGF_FFMEMLESS_SETTINGS=/usr/share/ngfd/plugins.d/ngf-vibra-DEVICE.ini
```

Now you can use those 2 files to tune force feedback effects suitable specifically for your device. For template to start making your own configuration files, just copy-paste the ngfd ffmemless.ini and Qtfeedback ffmemless.ini default config files as the device specific files and then edit only needed bits.

The reason we have possibility for device specific effects is that hardware mechanics and the vibra engines differ greatly device-by-device, and single settings will not give good effect on all devices.

- **At minimum, you should ALWAYS tune at least KEYPAD effect in qtfeedback-DEVICE.ini for every device separately to make the VKB haptic feel good and punctual.**

Good guideline for VKB haptic is that it should be as short as possible, and vibrate at the resonance frequency of the device mechanics when vibra engine reaches top magnitude of the vibra effect. It should not feel like vibration, but like a single kick.

14.4 Non-Graphical Feedback Daemon

The Non-Graphical Feedback Daemon provides combined audio, haptic, and LED feedback for system events and alarms. These events include such things as ring tones, message tones, clock alarms, email notifications, etc.

- https://git.merproject.org/mer-core/ngfd

TODO: add more detail about configuring NGFD.

14.5 Non-Graphic Feedback Daemon PulseAudio Plugin

TODO

14.6 Non-Graphic Feedback Daemon Droid ffmemless Plugin

This is the main plugin handling vibra feedback for Sailfish OS. See *Configuring haptics* for more details.

14.7 Non-Graphic Feedback Daemon Droid Vibrator Plugin

This is a secondary vibra plugin for demoing and quick ports. It works out of the box with android timed output drivers. The feature set is reduced compared to ffmemless plugin.

TODO

14.8 PulseAudio Droid Modules

TODO - more information about how PA works

14.9 Qt5 QtFeedback Droid Vibrator Plugin

TODO

14.10 Qt5 Hardware Composer QPA

This Qt Platform Abstraction plugin makes use of the libhardware hwcomposer API to send rendered frames from the Wayland Compositor to the actual framebuffer. While for some older devices, just flipping the fbdev was enough, more recent devices actually require using hwcomposer to request flipping and for vsync integration.

The important environment variables are:

- `EGL_PLATFORM`: For the Wayland Compositor, this needs to be set to `fbdev` on devices with older hwcomposer versions, and to `hwcomposer` for hwcomposer version 1.1 and newer. For best results, first try `fbdev`, and if it doesn't work, try `hwcomposer` instead. For the Wayland Clients, this always needs to be set to `wayland`.
- `QT_QPA_PLATFORM`: For the Wayland Compositor, this needs to be set to `hwcomposer` to use the plugin. Previously, `eglfs` was used, but the `hwcomposer` module replaces the old plugin on Sailfish OS on Droid. For Wayland Clients, this always needs to be set to `wayland`.

When starting up an application (e.g. the Wayland Compositor, `lipstick`), the systemd journal (`journalctl -fa` as user root) will show some details about the detected screen metrics, which will come from the framebuffer device:

```
HwComposerScreenInfo:251 - EGLFS: Screen Info
HwComposerScreenInfo:252 -  - Physical size: QSizeF(57, 100)
HwComposerScreenInfo:253 -  - Screen size: QSize(540, 960)
HwComposerScreenInfo:254 -  - Screen depth: 32
```

Also, it will print information about the hwcomposer module and the device. In this specific case, the hwcomposer version is 0.3:

```
== hwcomposer module ==
* Address: 0x40132000
* Module API Version: 2
* HAL API Version: 0
* Identifier: hwcomposer
* Name: Qualcomm Hardware Composer Module
* Author: CodeAurora Forum
== hwcomposer module ==
== hwcomposer device ==
* Version: 3 (interpreted as 30001)
* Module: 0x40132000
== hwcomposer device ==
```


The source tree contains different implementations of hwcomposer backends, each one for a different hwcomposer API version (see `hwcomposer/hwcomposer_backend.cpp`). Based on that detection, one of the existing implementations is used. Right now, the following implementations exist:

- *hwcomposer_backend_v0*: Version 0.x (e.g. 0.3) of the hwcomposer API. It can handle swapping of an EGL surface to the display, doesn't use any additional hardware layers at the moment and can support switching the screen off. The VSync period is queried from the hwcomposer device, but it will fall back to 60 Hz if the information cannot be determined via the libhardware APIs. (`EGL_PLATFORM=fbdev`)
- *hwcomposer_backend_v10*: Version 1.0 of the hwcomposer API. It supports one display device, handles VSync explicitly and uses a single hardware layer that will be drawn via EGL (and not composed via hwcomposer). Swapping is done by waiting for VSync and uses libsync-based synchronization of posting buffers. Switching the screen off is also supported, and sleeping the screen disables VSync events. Also, the same VSync period algorithm is used (try to query from libhardware, fall back to 60 Hz if detection fails). (`EGL_PLATFORM=fbdev`)
- *hwcomposer_backend_v11*: Version 1.1, 1.2, 1.3, 1.4, and 1.5 of the hwcomposer API. Versions higher or equal than 1.3 only support physical displays, whereas 1.1 and 1.2 support also virtual displays. This requires libsync and hwcomposer-egl from libhybris. Most of the hwcomposer 1.0 API properties apply, with the exception that frame posting and synchronization happens with the help of libhybris' hwcomposer EGL platform. (`EGL_PLATFORM=hwcomposer`)

Instead of running the Wayland Compositor (lipstick) on top of the hwcomposer QPA plugin, one can also run all other Qt 5-based applications, but the application can only open a single window (multiple windows are not supported, and will cause an application abort). For multiple windows, Wayland is used. This means that for testing, it is possible to run a simple, single-window Qt 5 application on the framebuffer (without any Wayland Compositor in between) by setting the environment variables `EGL_PLATFORM` and `QT_QPA_PLATFORM` according to the above.

14.11 SensorFW Qt 5 / libhybris Plugin

TODO

14.12 Build HA Middleware Packages

`rpm/dhd/helpers/build_packages.sh` now is taking care of builds/rebuilds/local repo preparation and patterns.

14.12.1 All other packages

Please compile any other required packages should a build/mic process indicate a dependency on them. Feel free to add/remove those packages to/from patterns to suit your port's needs.

Follow the exact same compilation approach as with above packages. Known packages are:

- https://github.com/mer-hybris/unblank-restart-sensors - needed only by mako

CHAPTER

FIFTEEN

LIST OF REPOSITORIES

droid-hal-$DEVICE Contains RPM packaging and conversion scripts for converting the results of the Android HAL build process to RPM packages and systemd configuration files.

hybris-boot Script run during Android HAL build that will combine the kernel and a custom initrd to `hybris-boot.img` and `hybris-recovery.img`. Those are used to boot a device into Sailfish OS and for development purposes.

hybris-installer Combines the `hybris-boot` output and the root filesystem into a .zip file that can be flashed via Android Recovery.

libhybris Library to allow access to Bionic-based libraries from a glibc-based host system (e.g. hwcomposer, EGL, GLESv2, ..).

qt5-qpa-hwcomposer-plugin Qt 5 Platform Abstraction Plugin that allows fullscreen rendering to the Droid-based hardware abstraction. It utilizes libhybris and the Android hwcomposer module.

mer-kernel-check A script that checks if the kernel configuration is suitable for Sailfish OS. Some features must be enabled, as they are needed on Sailfish OS (e.g. to support `systemd`), other features must be disabled, as they conflict with Sailfish OS (e.g. `CONFIG_ANDROID_PARANOID_NETWORK`) at the moment.

CHAPTER

SIXTEEN

PACKAGE NAMING POLICY

For consistency, certain hardware adaptation / middleware plugin packages have to be named after a certain pattern.

As in the other chapters of this guide, `$DEVICE` should be replaced with the device codename (e.g. `mako` for Nexus 4), and the asterisk (*) is used as wildcard / placeholder.

16.1 List of naming rules

Packages that are arch-specific (e.g. `armv7hl`), device-specific and contain `$DEVICE` in their name:

- The arch-specific HAL RPMs (built from `droid-hal-device`) should be named **droid-hal-$DEVICE** (e.g. `droid-hal-mako`, `droid-hal-mako-devel`, `droid-hal-mako-img-boot`, `droid-hal-mako-kernel`, `droid-hal-mako-kernel-modules`, `droid-hal-mako-kickstart-configuration`, `droid-hal-mako-patterns`, `droid-hal-mako-policy-settings` and `droid-hal-mako-pulseaudio-settings`)
- The package containing kickstart files for `mic` should be named **ssu-kickstarts-$DEVICE** (e.g. `ssu-kickstarts-mako`)

Package that are arch-independent (`noarch`), device-specific and contain `$DEVICE` in their name:

- The arch-independent HAL RPMs (built from `droid-hal-device`) should be named: **droid-hal-$DEVICE-*** (e.g. `droid-hal-mako-img-recovery` and `droid-hal-mako-sailfish-config`)
- The SensorFW libhybris plugin configuration should be named **hybris-libsensorfw-qt5-configs** (`hybris-libsensorfw-qt5-configs`)

Packages that are arch-specific (e.g. `armv7hl`), device-specific, but do not contain `$DEVICE`:

- RPMs built from libhybris should be named **libhybris-*** (e.g. `libhybris-libEGL`)
- Plugins for the non-graphic feedback daemon should be named **ngfd-plugin-*** (e.g. `ngfd-plugin-droid-vibrator`); as well as their Qt plugin **qt5-feedback-haptics-droid-vibrator** (`qt5-feedback-haptics-droid-vibrator`)
- The QPA hwcomposer plugin should be named **qt5-qpa-hwcomposer-plugin** (`qt5-qpa-hwcomposer-plugin`)
- The PulseAudio support modules should be named **pulseaudio-modules-droid** (`pulseaudio-modules-droid`)
- The GStreamer plugins should be named **libgstreamer0.10-*** and/or **gstramer0.10-*** (e.g. `libgstreamer0.10-gralloc`, `libgstreamer0.10-nativebuffer`, `gstreamer0.10-omx`, `gstreamer0.10-droideglsink` and `gstreamer0.10-droidcamsrc`)
- The SensorFW libhybris plugin should be named **hybris-libsensorfw-qt5** (`hybris-libsensorfw-qt5`)

16.2 List of Provides

- **droid-hal-$DEVICE-*** provides **droid-hal-*** (e.g. `droid-hal-$DEVICE-pulseaudio-settings` provides `droid-hal-pulseaudio-settings`)

16.3 TODO

The above "rules" are the current state of our hardware adaptation. Here are some things that should be improved there:

- Some arch-specific packages contain arch-independent config files or binary blobs - make them arch-independent (`noarch`) instead
- Unify the GStreamer plugin naming (either **libgstreamer0.10-*** or **gstreamer0.10-***) to not have two naming schemes there
- The PulseAudio settings package usually is called **pulseaudio-settings-$DEVICE** (we currently have **droid-hal-$DEVICE-pulseaudio-settings**, maybe this can be implemented as a `Provides:`?)
- The Linux kernel modules are in **droid-hal-$DEVICE-kernel-modules** at the moment, in other hardware adaptations we use **kmod-xyz-$DEVICE**
- The recovery partition in the image at the moment is **droid-hal-$DEVICE-img-recovery**, but for other hardware adaptations we use **jolla-recovery-$DEVICE**

CHAPTER SEVENTEEN

LICENSE

Creative Commons Legal Code

Attribution-NonCommercial-ShareAlike 3.0 Unported

> CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(g) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. "Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

d. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.

e. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

f. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

g. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

h. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

i. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

j. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

d. to Distribute and Publicly Perform Adaptations.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights described in Section 4(e).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(d), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(d), as requested.

b. You may Distribute or Publicly Perform an Adaptation only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-NonCommercial-ShareAlike 3.0 US) ("Applicable License"). You must include a copy of, or the URI, for Applicable License with every copy of each Adaptation You Distribute or Publicly Perform. You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in con-nection with the exchange of copyrighted works.

d. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(d) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original

Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

e. For the avoidance of doubt:

   i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

   ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c).

f. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING AND TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THIS EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

> Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.
>
> Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.
>
> Creative Commons may be contacted at http://creativecommons.org/.