Hardware

- Flashing the Jolla C2 using the Unisoc flash mode
- USB audio DAC with Udev and Pulse
- HADK
  - HADK
  - HADK - Hot

Flashing the Jolla C2 using the Unisoc flash mode

This information was originally in a forum post: https://forum.sailfishos.org/t/flashing-the-jolla-c2/21020/3

The Jolla C2 contains a Unisoc UMS9230 (Tiger T606) SoC. All Unisoc SoCs implement a special USB flashing protocol in their boot ROM, which can be entered by pressing the Volume Up and Power keys. This provides a way to reflash the phone even if a broken bootloader was flashed.

This flashing interface is usually used with Unisoc's flashing software to flash PAC files containing the entire factory image for the phone and a loader signed with device-specific keys. However, the signature check can be bypassed on this SoC, and tools have been developed to allow flashing partitions directly with an unsigned loader.

Initial Setup

Follow these steps to set up your environment for flashing the Jolla C2:

1. Download https://github.com/TomKing062/spreadtrum_flash and build it using make (for Windows users, a prebuilt .exe is available in the ums9230 unlock archive, but please note that the examples below assume a Unix-like system)
2. For Linux users: set up udev rules if you want to run the flashing commands with normal user permissions. See this README for more information.
3. Download the ums9230_universal_unlock.zip archive from here and extract it into the main spreadtrum_flash directory (required files are `custom_exec_no_verify_65015f08.bin`, `fdl1-dl.bin` and `fdl2-dl.bin`)
4. Obtain a bootloader image for the Jolla C2 / Reeder S19 Max Pro S, from one of the following places:
   - from an archive with images: the file will be named `uboot_a.img`, `uboot_a.bin` or `lk.bin`
   - **or** from the phone directly by using `fdl2-dl.bin` temporarily (make sure to read about the side effects in the "Troubleshooting" section):

```
./spd_dump \
    exec_addr 0x65015f08 \
    fdl fdl1-dl.bin 0x65000800 \
    fdl fdl2-dl.bin 0x9efffe00 exec \
    r uboot_a reset
```

Flashing partitions

To flash partitions to the device, use the following command:

```
./spd_dump \
    exec_addr 0x65015f08 \
    fdl fdl1-dl.bin 0x65000800 \
    fdl <path to your FDL2 file> 0x9efffe00 exec \
    w <partition name> <image path> \ # This line can be repeated for multiple partitions
    reset
```

It is recommended to use the device-specific bootloader as the FDL2 unless you are flashing a critical bootloader partition (splloader or uboot). The device-specific bootloader does not allow flashing these partitions, while fdl2-dl.bin from the exploit package does. See the "Troubleshooting" section for the expected side effects of using it.

Flashing Sailfish OS

This should also work with a Reeder S19 Max Pro S: see https://forum.sailfishos.org/t/installing-sailfish-to-reeder-s19-max-pro-s/21873/20

Get the Sailfish OS images from https://releases.sailfishos.org/images/. Assuming they are unpacked into a directory named SailfishOS, run these commands to flash them:

```
simg2img SailfishOS/sailfish.img001 sailfish.img
simg2img SailfishOS/fimage.img001 fimage.img
./spd_dump \
    exec_addr 0x65015f08 \
    fdl fdl1-dl.bin 0x65000800 \
    fdl fdl2-dl.bin 0x9efffe00 exec \
    w uboot_a SailfishOS/lk.bin \
    w uboot_b SailfishOS/lk.bin \
    w dtbo_a SailfishOS/dtbo.img \
    w dtbo_b SailfishOS/dtbo.img \
    reset
# Hold down the volume up key while the phone reboots!
./spd_dump \
    exec_addr 0x65015f08 \
    fdl fdl1-dl.bin 0x65000800 \
    fdl SailfishOS/lk.bin 0x9efffe00 exec \
    w boot_a SailfishOS/hybris-boot.img \
    w boot_b SailfishOS/hybris-boot.img \
```

```
w vendor_boot_a SailfishOS/vendor_boot.img \
w vendor_boot_b SailfishOS/vendor_boot.img \
w userdata sailfish.img \
w super fimage.img \
reset
```

Troubleshooting

The flashing tool cannot connect to the phone:
Make sure to boot into flashing mode using Volume Up + Power. You should not see anything on the screen!

The command fails with timeout or connection errors:
Restart the flashing mode by holding the Volume Up and Power keys for 7 seconds. This is usually required if a command is interrupted.

The phone reboots when the Volume Down key is pressed!
This is a **side effect** of using `fdl2-dl.bin`. This behavior disappears after the phone is shut down completely.

The phone reboots while flashing or dumping large partitions:
This is typically also caused by `fdl2-dl.bin`. Use the device-specific bootloader instead.

USB audio DAC with Udev and Pulse

This is a brief description of how to get various Usb Audio DACs (dongle with usb c -> 3.5mm audio jack). Running. Based on https://www.oesf.org/forum/index.php?topic=35327.0

To use a USB DAC (digital-analog audio converter) in Sailfish OS it is necessary to set up udev to respond to the device being plugged in and turn off the internal sound card.

Some devices like the Fairphone 4/5 (possibly the xperia 10iii) may already be working without issue.

On an xperia 10iii or a Volla22, as of January 2026, you need to become root and follow these steps:

1. become root

```
devel-su
```

2. check or modify the file /etc/pulse/arm_droid_card_custom.pa to contain

```
load-module module-droid-card rate=48000 hw_volume=false unload_call_exit=true output_voip_rx=false
record_voice_16k=true
load-module module-udev-detect
```

3. make sure pulse audio has usb audio configured with the file /etc/pulse/xpolicy.conf.d/usbaudio.conf By default it is already available but disabled. It should be symlinked as follows:

```
cd /etc/pulse/xpolicy.conf.d/
ln -s usbaudio.conf.disabled usbaudio.conf
```

That's it unless your device does not have a udev rule at: /lib/udev/rules.d/99-pulseaudio-ignore-builtin-card.rules

All currently supported official Sailfish OS devices have the udev rule needed for USB audio. The Volla22 does not. To make one:

1. Find the device id for the internal audio card:

```
devel-su
cat /sys/class/sound/card0/id
mt6768mt6358 # volla22
```

2. Create a udev rule file:

either with a single commend (echo places the text betwee ' ' into > the file /.....)

```
echo 'KERNEL=="card0", SUBSYSTEM=="sound", ATTR{id}=="mt6768mt6358", ENV{PULSE_IGNORE}="1"' > /etc/udev/rules.d/89-pulseaudio-usb.rules
```

Rules for other cases (ie. xperia 10ii).

as root:

```
devel-su
```

modify the udev rules file

```
vi /etc/udev/rules.d/89-pulseaudio-usb.rules
```

with the following

```
ACTION=="add", SUBSYSTEM=="sound", RUN+="/usr/local/bin/udev-create-flag"
ACTION=="change", SUBSYSTEM=="sound", RUN+="/usr/local/bin/udev-create-flag"
```

Create a flag file

```
vi /usr/local/bin/udev-create-flag
```

with contents

```
#!/bin/sh
# called by udev (runs as root)
# write a flag in the user's runtime dir so the user service notices it
USER="defaultuser"
FLAG="/run/user/$(id -u "$USER")/usb-dac-connected.flag"
mkdir -p "$(dirname "$FLAG")"
touch "$FLAG"
```

make it executable and create the script to trigger the dac (or set volume, for instance)

```
chmod 755 /usr/local/bin/udev-create-flag
```

```
vi /usr/local/bin/set-usb-dac-volume.sh
```

with content this version is NOT setting volume, just making sure the dac is seen

```
#!/bin/sh
sleep 2
/usr/bin/pactl load-module module-udev-detect
```

or for also setting volume (apple dongle for instance) add

```
# !!! update sink name to yours - it contains unique serial number !!! `pacmd list-sinks|grep alsa_output.usb`
# someone could improve this to autotdetec script
sleep 2
/usr/bin/pactl load-module module-udev-detect
pactl set-sink-volume alsa_output.usb-Apple__Inc._USB-
C_to_3.5mm_Headphone_Jack_Adapter_DWH504202DRL1MQAF-00.analog-stereo 100%
```

make it executable

```
 chmod 755 /usr/local/bin/set-usb-dac-volume.sh
```

Now create the systemd triggers (these are prompted by the udev create flag script)

you may need to make the directories first:

```
mkdir /home/defaultuser/.config/systemd/user
```

```
vi /home/defaultuser/.config/systemd/user/usb-dac-volume.path
```

with contents

```
[Unit]
Description=Watch for USB DAC flag
[Path]
PathExists=/run/user/%U/usb-dac-connected.flag
```

```
Unit=usb-dac-volume.service
[Install]
WantedBy=default.target
```

```
vi /home/defaultuser/.config/systemd/user/usb-dac-volume.service
```

with contents

```
[Unit]
Description=Run USB DAC volume script when flag present
After=default.target
[Service]
Type=oneshot
Environment=XDG_RUNTIME_DIR=%t
ExecStart=/usr/local/bin/set-usb-dac-volume.sh
ExecStartPost=/bin/rm -f /run/user/%U/usb-dac-connected.flag
TimeoutStartSec=60
[Install]
WantedBy=default.target
```

and enable

```
systemctl --user daemon-reload
systemctl --user enable --now usb-dac-volume.path
```

You should now have automatic switching between internal and usb dacs.

Article discussion: [USB DAC playback on Sailfish OS Forum]

HADK

This chapter contains:
* the SFOS hardware porters HADK fag
* SFOS hardware porters documentation

HADK

1 Overview

v4.5.0.19 Sailfish OS Hardware Adaptation Development Kit Documentation

Copyright 2014-2022 Jolla Ltd. | Content licensed under CC-BY-NC-SA 3.0 Unported

This page contains the HADK contents derived from:
https://docs.sailfishos.org/Develop/HADK/SailfishOS-HardwareAdaptationDevelopmentKit-4.5.0.19.pdf

Preparations

1.1 Goal

By following this guide you can set up a Sailï¬•sh OS (or another Sailï¬•sh 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:
â€¢ Sailï¬•sh Core : the GNU/Linux userspace core
â€¢ Android Hardware Adaptation (HA/HAL), consisting of:
â€¢ Device-speciï¬•c Android Kernel
â€¢ Android base which can be:
â€¢ LineageOS - https://wiki.lineageos.org
â€¢ AOSP - Android Open Source Project - https://source.android.com
â€¢ CAF - Code Aurora Forum - https://www.codeaurora.org
â€¢ Sony Open Devices program - https://developer.sony.com/develop/open-devices
â€¢ Vendor-speciï¬•c Android base
â€¢ Binary device drivers taken from an Android base
â€¢ Hybris patches to the Android base
â€¢ The libhybris interface built against the binary drivers
â€¢ Middleware packages depending on hardware-speciï¬•c plugins
â€¢ A Qt/Wayland QPA plugin utilizing the Android `hwcomposer`
â€¢ Sailï¬•sh OS components

Instructions

1.2 Development

1.2.1 Requirements

The development environment uses the Platform SDK, with:
â€¢ Build Tools consisting of cross-compilers (tooling) and an emulated rootfs for your device architecture (tar-
get), containing device-specific headers and libraries â€“ will also be referred as build environment through-
out the document
â€¢ a HA build SDK (a minimal Ubuntu chroot required to build the Android sources

During the HA development youâ€™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 build environment 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, Build Tools, and Ubuntu chroot with the environment variable $PLATFORM_SDK_ROOT. You need around 10GB 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.
â€¢ In the HA build SDK
â€¢ a kernel
â€¢ a hacking friendly initrd which supports various boot options
â€¢ hybris-boot.img and hybris-recovery.img (for booting and debugging)
â€¢ a minimal Android /system/ tree
â€¢ modified Android parts for compatibility with libhybris and Sailfish OS (e.g. Bionic libc, logcat, init, . . .)
â€¢ In the Platform SDK
â€¢ RPM packages containing all the built binaries and extracted configs
â€¢ 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.

2 PREREQUISITES

2.1 Mobile Device

â€¢ An Android device officially supported by LineageOS 15.1 (Android 8), 16.0 (Android 9) and 17.1 (Android 10) at the time of writing 2021-02-12. CyanogenMod versions (that are Sailfish OS-compatible) 10.1.x, 11.0, 12.1, 13.0, 14.1 will require additional effort because CM has become obsolete. For more supported Android versions also check this link
â€¢ Throughout this guide we shall use the term Android base, which will refer to the appropriate base that you are porting on: LineageOS, AOSP, CAF etc
â€¢ We also support Sony Open Devices program, and published guidelines how to rebuild flashable images for:
â€¢ Xperia X (Sony AOSP 6)
â€¢ Xperia XA2 (Sony AOSP 8)
â€¢ Xperia 10 (Sony AOSP 9)
â€¢ Xperia 10 II (Sony AOSP 10)

â€¢ Xperia 10 III (Sony AOSP 11)
â€¢ Starting with Sailfish OS 3.4.0, support for 64bit ARM SFOS userspace has been added
â€¢ Sailfish OS adaptations starting with CM 13.0 (Android 6) were constructed by running a mix of 64bit Linux Kernel and Android HAL, whilst Sailfish OS userspace was being run in the 32bit mode. Such mixed mode is still supported, but we encourage porters to switch to full 64bit ports (using Xperia 10 II as reference)
â€¢ See https://wiki.lineageos.org/devices for a list of compatible devices
â€¢ See https://wiki.merproject.org/wiki/Adaptations/libhybris for a status list of devices already ported using HADK
â€¢ 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
â€¢ AOSP or CAF Android base support is also possible, but we choose 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)
â€¢ 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

â€¢ A 64-bit x86 machine with a 64-bit Linux kernel
â€¢ Sailfish OS Platform SDK (installation explained later)
â€¢ At least 30 GiB of free disk space (20 GiB source download + more for building) for a complete Android
6 build; a minimal download and HADK build (only hardware adaptation-related components) requires
slightly less space. Newer Android base versions yield increasingly bigger size requirements.
â€¢ At least 4 GiB of RAM (the more the better)

3 PREPARING YOUR DEVICE

3.1 Backup and Verify Your Device

As mentioned above, it might be helpful to backup the existing stock Android image before flashing the Android
base release for the first time, as obtaining 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 Android Recovery (e.g. TWRP or ClockworkMod) 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), itâ€™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 your Android base image

Flash an image that you built or obtained of your Android base, whether itâ€™s LineageOS, CAF, AOSP, or another.
The official LineageOS flashing instructions can be found on this LineageOS wiki page.

You may also want to verify that the Android base 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 Sailfish OS build results with.

You should at least check the following features:

â€¢ OpenGL ES 2.0: Use e.g. Gears for Android to test (the hz you will get there will be max refresh rate).
â€¢ WLAN connectivity: Connect to an AP, ad-hoc or set up a mobile access point with your device.
â€¢ Audio: Headset detection, earpiece speaker, loudspeakers, etc.
â€¢ Bluetooth: Connect to bluetooth headsets, verify discoverability, send files.
â€¢ NFC: Check if NFC tags can be detected, read and/or written by the device.
â€¢ SD/MicroSD: Use a file manager app to see if inserted SD cards can be detected.
â€¢ USB: MTP, mass storage (if available) and adb access.
â€¢ Telephony: 2G/3G/LTE calls + data connectivity.

â€¢ GPS: Using GPS Test, check GLONASS too; typical time to fix; AGPS.
â€¢ Sensors: Using AndroSensor: Accelerometer, Proximity Sensor, Ambient Light Sensor, Gyroscope, Mag-
netometer (Compass), Hall (flip case), . . .
â€¢ LEDs: If your device has notification LEDs or keypad backlights.
â€¢ Camera (front and back): Also test functionality of zoom, flash, etc..
â€¢ Buttons: Volume up, volume down, power, camera shutter, etc..
â€¢ Video out: HDMI / MHL connectivity if you have the necessary adapters. TV out.
â€¢ Screen backlight: Suspend and backlight control, minimum and maximum brightness.
â€¢ Battery meter: Charge level, battery health, charging via USB (wall charger and host PC).
â€¢ Vibration motor: Intensity, patterns.
â€¢ HW composer version: check dumpsys SurfaceFlinger through ADB (see SF Layer Debugging).
â€¢ Fingerprint sensor
â€¢ FM Radio

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

4 SETTING UP THE SDKS

4.1 Setting up required environment variables

Throughout this guide we will be referencing the location of your SDK, device build environment 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, weâ€™ll use
Nexus 5 (lge/hammerhead for its vendor/device pair), and port it using CyanogenMod 11.0 version as the
â€œAndroid baseâ€. Thus ensure you read the code 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' > $HOME/.hadk.env
export ANDROID_ROOT="$HOME/hadk"
export VENDOR="lge"
export DEVICE="hammerhead"
# "armv7hl" is still supported, but we encourage to have full 64bit ports
export PORT_ARCH="aarch64"
# Uncomment the next line to conveniently build all RPMs in local repo:
#alias mb2='mb2 --output-dir "${ANDROID_ROOT?}/droid-local-repo/${DEVICE?}"'
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 docs (â€œQuick startâ€ section is enough, do not install SDK Targets yet):

https://docs.sailfishos.org/Tools/Platform_SDK/Installation/

Afterwards, temporarily leave the PLATFORM_SDK to top up your ~/.bashrc with necessary commands:

```
PLATFORM_SDK $
exit

HOST $
cat <<'EOF' >> $HOME/.bashrc
if [[ $SAILFISH_SDK ]]; then
function hadk() { source $HOME/.hadk.env; echo "Env setup for $DEVICE"; }
```

```
hadk
fi
EOF
sfossdk
```

> **Warning**: With Platform SDK version 4.4.0.58 and olders you need to check the MERSDK variable instead
> of SAILFISH_SDK in the above code snippet.

Youâ€™ll need some tools which are not installed into the Platform SDK by default:
â€¢ android-tools-hadk contains tools and utilities needed for working with the Android SDK
â€¢ kmod is needed by micâ€™s qemu to build the image
â€¢ createrepo_c is needed when passing local repo to mic

```
PLATFORM_SDK $
sudo zypper ref
sudo zypper in android-tools-hadk kmod createrepo_c
```

The minimum Platform SDK SFOS version is 4.3.0.15. Use sdk-assistant command to upgrade your build
tools, 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
```

More information about keeping your SDK up-to-date: https://github.com/sailfishos/sdk-setup/blob/master/
sdk-setup/README.tips.wiki#SDK_Maintenance

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. For Android device ports that require OpenJDK 1.8 or newer, the following
commands download and unpack the rootfs to the appropriate location:

```
PLATFORM_SDK $
TARBALL=ubuntu-focal-20210531-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
```

In case you find youâ€™re not able to gain sudo privileges inside the Ubuntu Chroot, execute the following inside
the Platform SDK:

```
PLATFORM_SDK $
sudo chroot $UBUNTU_CHROOT /bin/bash -c "chage -M 999999 $(id -nu 1000)"
```

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
HABUILD_SDK $
# Now you are in the HABUILD_SDK environment
# To leave, just type `exit` or Ctrl+D, and you'll be back to the PLATFORM_SDK
```

4.3.3 If your port requires OpenJDK 1.7 or older

Our ubu-chroot environment is based on 20.04 LTS which provides OpenJDK 1.8 or newer.

If your Android base build requires an older Java Development Kit, please install the legacy ubu-chroot instead:

```
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
```

5 BUILDING THE ANDROID HAL

5.1 Checking out Source of the Android base

Our build process is based around the Android source tree, but where needed weâ€™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:

```
HABUILD_SDK $
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
Ensure Ubuntu chroot has cpio installed:
HABUILD_SDK $
sudo apt-get install cpio
```

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

Note: If your port requires OpenJDK 1.7 or older, use the older repo tool for legacy

Python 2 systems.

After youâ€™ve installed the repo command, a set of commands below will download the required projects for
building the modified parts of the Android base used in Sailfish OS hardware adaptations.

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

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

Alternatively, you can patch an Android base of your choosing (e.g. be it CAF or AOSP or another).

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

If your primary ROM does not match your Android base or its version, and you would like to keep it on your
device, then look for MultiROM support for it. Starting with its version v28, it supports booting Sailfish OS.

This porting guide is using Nexus 5 and CyanogenMod 11.0 version as example:

```
HABUILD_SDK $
sudo mkdir -p $ANDROID_ROOT
sudo chown -R $USER $ANDROID_ROOT
cd $ANDROID_ROOT
repo init -u https://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, youâ€™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 in the LineageOS device wiki, for Nexus
5 it would be https://wiki.lineageos.org/devices/hammerhead/build#initialize-the-lineageos-source-repository Lo-
cal 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: Do not use â€“fetch-submodules parameter on hybris-
18.1 or newer Android bases.

```
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, we cannot use generic partition names (independent of partition number) for example: /dev/block/bootdevice/by-name/userdata. These are symlinks to real block device node /dev/mmcblkXpY or /dev/sd*X created by udev but it starts after initrd. In initrd we then have to specify hardcoded block device nodes for /boot and /data partitions instead convenient by-name nodes.

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 â€“ all done by $ANDROID_ROOT/rpm (droid-hal-device).

Unfortunately, systemd cannot recognise named partition (by-name) 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 script for each device, for all partitions â€“ in this way we are sure to cover them all, because if done manually by looking through fstab/rc files, some might get unnoticed.

fixup-mountpoints will convert by-name nodes paths defined in android's fstab and .rc files (for example: /dev/block/platform/msm_sdcc.1/by-name/system) to block device (for example: /dev/mmcblk0p23) which is needed to create working systemd mount units: /usr/lib/systemd/system/*.mount. They will take care of mounting your partitions before droid-hal-init starts. Therefore, to create correct map (entry in fixup-mountpoints) for your device you need by-name paths from your fstab (find it in $ANDROID_ROOT/device/$VENDOR/*/rootdir/ or directly on device) for example:

https://github.com/LineageOS/android_device_fxtec_pro1/blob/5e8025c5bc6958e2f6319fbb82b43e16552617b0/rootdir/etc/fstab.qcom#L12. Notice how /userdata and /system use slightly different by-name path for this particular device so do not assume that your device use "block/bootdevice/by-name/" or "block/platform/msm_sdcc.1/by-name" path for all partitions, check fstab instead! Another thing worth to notice are entires without by-name paths like "system /system" clearly indicating dynamic partitions. Also do not follow _a/_b syntax in fixup-mountpoints used for official devices like pdx213 unless you really have these suffixes in lineage's fstab.

Then you need list of all partitions with coresponding real block device nodes. To get that you should flash and boot an image of your Android base and execute adb shell on your host and something like this:

```
ls -l /dev/block/platform/*/by-name/
or /dev/block/platform/*/*/by-name/
```

then add mapping to fixup-mountpoints. For example: https://github.com/mer-hybris/hybris-boot/pull/134/files

If your device use system-as-root then follow also: https://sailfishos.wiki/link/20#bkmrk-%C2%A0system-as-root%3A-htt

If android's fstab do not include '/system' (or '/' in case of system-as-root), '/vendor' or '/boot' entry then you might need to add them manually.

Once youâ€™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 donâ€™t forget to upstream your fixup-mountpoints
patch.

5.4 Building Relevant Bits of your Android base

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).

Youâ€™ll probably need to iterate this a few times to spot missing repositories, tools, configuration files and others:
Before building it is recommended to read extra Android base specific hints from
https://github.com/mer-hybris/
hadk-faq#android-base-specific-fixes

```
HABUILD_SDK $
source build/envsetup.sh
export USE_CCACHE=1
breakfast $DEVICE
make -j$(nproc --all) hybris-hal droidmedia
```

The relevant output bits will be in out/target/product/$DEVICE/, in particular:
â€¢ hybris-boot.img: Kernel and initrd

â€¢ hybris-recovery.img: Recovery boot image
â€¢ system/ and root/: HAL system libraries and binaries

The approximate size of the output directory out/ after make hybris-hal is 10 GB (as of 2019-03-14, hybris-sony-aosp-8.1.0_r52-20190206 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 your Android base is using. Which one? Itâ€™s different for
every device, most likely first:

â€¢ Check the value of TARGET_KERNEL_CONFIG under
$ANDROID_ROOT/device/$VENDOR/*/BoardConfig*.mk
â€¢ Examine the output of make bootimage for which defconfig is taken when youâ€™re building kernel, e.g.:
make -C kernel/lge/hammerhead ... cyanogenmod_hammerhead_defconfig
â€¢ Check your Android base kernelâ€™s commit history for the arch/arm*/configs folder, look for def-
config

If you are in a rush, get rid only of ERROR cases first, but donâ€™t forget to come back to the WARNING ones too.

After youâ€™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 de-
vice/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 isnâ€™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

â€¢ 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

â€¢ If repo sync --fetch-submodules fails with some other error message try running repo sync
to see if it helps. This is usually needed for hybris-18.1 or newer Android bases.

â€¢ If you notice git clone commands starting to write out â€œForbidden . . . â€ 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

â€¢ error: Cannot fetch . . . (GitError: â€“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
```

â€¢ In some cases (with parallel builds), the build can fail, in this case, use make -j1 ... 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
```

6 INSTALLING BUILD TOOLS FOR YOUR DEVICE

It is necessary to emulate your target device architecture and file system to build hardware adaptation packages in
the next section. Download and install your build tools following instructions below.

> **Important: Minimum version for SFOS target is 4.3.0.15 (same requirement as for the Platform SDK Chroot
> earlier)**

> **Warning**: To ensure consistency with HADK build scripts, name your tooling SailfishOS-4.5.0 (or
> whichever release you are building for) instead of suggested SailfishOS-latest, and your target as
> $VENDOR-$DEVICE-$PORT_ARCH (instead of SailfishOS-latest-aarch64). Ignore the i486 target.

For ARM devices, choose aarch64 build target, unless you are building for the armv7hl Sailfish OS userspace.

https://docs.sailfishos.org/Tools/Platform_SDK/Target_Installation/

To verify the correct installation of the build tools, cross-compile a simple â€œHello, World!â€ C application with
mb2 build-shell:

```
PLATFORM_SDK $
cd $HOME
mkdir hadk-test-tmp
cd hadk-test-tmp



cat > main.c << EOF
#include <stdlib.h>
#include <stdio.h>
int main(void) {
    printf("Hello, world!\n");
    return EXIT_SUCCESS;
}
EOF
mb2 -t $VENDOR-$DEVICE-$PORT_ARCH build-init
mb2 -t $VENDOR-$DEVICE-$PORT_ARCH build-shell 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 mb2 setup):

```
mb2 -t $VENDOR-$DEVICE-$PORT_ARCH build-shell ./test
```

The above command should output â€œHello, world!â€ on the console, this proves that the build tools can compile
binaries and execute them for your architecture.

7 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 with hammerhead codename from lge vendor, adjust as appropriate and push to your GitHub home):

```
PLATFORM_SDK $
cd $ANDROID_ROOT
# Make sure $DEVICE and $VENDOR env variables are set correctly
# For Nexus 5 this should prints "hammerhead lge"
echo "$DEVICE $VENDOR"
mkdir rpm
cd rpm
git init
git submodule add https://github.com/mer-hybris/droid-hal-device dhd
# Rename 'hammerhead' and other values as appropriate
sed -e "s/@DEVICE@/$DEVICE/" \
-e "s/@VENDOR@/$VENDOR/" \
-e "s/@DEVICE_PRETTY@/Nexus 5/" \
-e "s/@VENDOR_PRETTY@/LG/" \
dhd/droid-hal-@DEVICE@.spec.template > droid-hal-$DEVICE.spec
# Please review droid-hal-$DEVICE.spec before committing!
git add .
git commit -m "[dhd] Initial content"
# Create this repository under your GitHub home
git remote add myname https://github.com/myname/droid-hal-$DEVICE.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@/$DEVICE/" \
-e "s/@VENDOR@/$VENDOR/" \
-e "s/@DEVICE_PRETTY@/Nexus 5/" \
-e "s/@VENDOR_PRETTY@/LG/" \
droid-configs-device/droid-config-@DEVICE@.spec.template > \
rpm/droid-config-$DEVICE.spec
# Please review rpm/droid-config-$DEVICE.spec before committing!
# Add tmp files to .gitignore
cat <<'EOF' >> .gitignore
tmp
documentation.list
debug*.list
installroot
EOF
git add .
git commit -m "[dcd] Initial content"
# Create this repository under your GitHub home
git remote add myname https://github.com/myname/droid-config-$DEVICE.git
git push myname master
cd -
rpm/dhd/helpers/add_new_device.sh
# On Nexus 5 the output of the last command is:
# Creating the following nodes:
# sparse/
# patterns/
# patterns/patterns-sailfish-device-configuration-$DEVICE.inc
# patterns/patterns-sailfish-device-adaptation-$DEVICE.inc
cd hybris/droid-configs
git add .
git commit -m "[dcd] Patterns and compositor config"
git push myname master
cd -
mkdir -p hybris/droid-hal-version-$DEVICE
cd hybris/droid-hal-version-$DEVICE
git init
git submodule add https://github.com/mer-hybris/droid-hal-version
mkdir rpm
sed -e "s/@DEVICE@/$DEVICE/" \
-e "s/@VENDOR@/$VENDOR/" \
-e "s/@DEVICE_PRETTY@/Nexus 5/" \
```

```
-e "s/@VENDOR_PRETTY@/LG/" \
droid-hal-version/droid-hal-version-@DEVICE@.spec.template > \
rpm/droid-hal-version-$DEVICE.spec
# Please review rpm/droid-hal-version-hammerhead.spec before committing!
git add .
git commit -m "[dvd] Initial content"
# Create this repository under your GitHub home
git remote add myname \
https://github.com/myname/droid-hal-version-$DEVICE.git
git push myname master
```

Since android 10 you must define android_version_major in droid-config. For example for android 10/lineage-17 add following line to droid-config-$DEVICE.spec:

```
%define android_version_major 10
```

All defines must be add before "%include droid-configs-device/droid-configs.inc"

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/$DEVICE.xml

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

It's good idea to create https://github.com/myname/local_manifests and add .repo/local_manifests/$DEVICE.xml simillar to how it's done in: 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

Before building the packages it is recommended to read extra Android base specific hints from https://github.com/
mer-hybris/hadk-faq#android-base-specific-fixes
The next step has to be carried out in the Platform SDK chroot:

```
PLATFORM_SDK $
cd $ANDROID_ROOT
rpm/dhd/helpers/build_packages.sh --droid-hal
rpm/dhd/helpers/build_packages.sh --configs
rpm/dhd/helpers/build_packages.sh --mw
rpm/dhd/helpers/build_packages.sh --gg
rpm/dhd/helpers/build_packages.sh --version
```

This will compile all the needed packages, patterns, middleware and put them under local repository. If anything
gets modified, just re-run the appropriate part.

7.2.2 Troubleshoot errors from build_packages.sh

â€¢ 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}
```

â€¢ Lastly, re-run build_packages.sh --droid-hal

8 CREATING THE SAILFISH OS ROOT FILESYSTEM

8.1 Additional Packages for Hardware Adaptation

See Middleware for a list of all middleware components (not all middleware components are used by every device
adaptation). Most of them will have already been built by the build_packages.sh --mw script, but if you
need an extra one, rebuild with 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 are not allowing to redistribute them. Examples:
â€¢ jolla-xt9 (predictive text input)
â€¢ sailfish-eas (Microsoft Exchange support)
â€¢ aliendalvik (AndroidTM App Support)
â€¢ sailfish-maps
â€¢ Any non-free audio/video codecs, etc.

8.3 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.

8.3.1 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
```

8.4 Building the Image with MIC

You need to choose a Sailfish OS version you want to build.

> Important: Avoid building older releases unless you know what youâ€™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 Installing Build Tools for Your Device. E.g., if targetâ€™s ssu
> lr versions begin with 4.5.0., build Sailfish OS update 4.5.0.19 (check for the latest, non â€œEarly Accessâ€
> 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)
export RELEASE=4.5.0.19
# EXTRA_NAME adds your custom tag. It doesn't support '.' dots in it!
export EXTRA_NAME=-my1
rpm/dhd/helpers/build_packages.sh --mic
```

Once obtained the .zip file, sideload via your deviceâ€™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 oftc.net about
details.

If creation fails due to absence of a package required by pattern, note down the package name.

If that package is critical (e.g. libhybris, qt5-qpa-hwcomposer-plugin etc.), build and add it to the local repo as explained in Additional Packages for Hardware Adaptation. Afterwards perform:
â€¢ Modifying a pattern
â€¢ 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:
â€¢ Modifying a pattern
â€¢ Building the Image with MIC

Alternatively (or if you canâ€™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 Troubleshooting

8.5.1 /dev/null - Permission denied (while using mic)

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

mount rules.

8.5.2 Executing commands in the build environment

You can execute commands to build and install packages under the build environment, inspect and debug any
issues. The syntax is shown in Installing Build Tools for Your Device.

Note that mb2 uses a working copy of your original build target, which means you can experiment with mb2
build-shell at will, but once you have found a desired fix, make it permanent by recording the changes in
your source code (e.g. do not leave installed packages with zypper in lying around, but add them to your
.specâ€™s BuildRequires).

If you break your build environment via mb2 build-shell, you can reset it back to its clean state via mb2 -t
$VENDOR-$DEVICE-$PORT_ARCH build-requires reset. This happens implicitly after re-running build_packages.sh1 .

Use

```
mb2 ... build-requires diff
```

if you want to know what you have done to your build environment
with mb2 in terms of installed/removed packages2 .

```
mb2 ... build-shell
```

is limited to launch only from directories where you previously ran commands like
mb2 ... build or mb2 ... build-init3 . Such commands are run under $ANDROID_ROOT during the build of dhd, so you can run mb2 build-shell from $ANDROID_ROOT if you find no better place.

1 As long as your original build target does not change, mb2 keeps using the same working copy (â€œsnapshotâ€ in mb2â€™s speech) of your build target in subsequent executions, preserving any changes you make to it. When your original build target changes, mb2 will reset the working copy to match the updated state of your original target next time it is invoked. This happens e.g. when you use build_packages.sh, which intentionally works directly on your original build target. Factors that are regarded as a change in the original build target are: RPM DB change, SSU configuration, and few other things.
2 If you need to make permanent changes to the original build environment (not recommended),

add --no-snapshot=force option at the beginning of mb2 command line (it is a global option).
3 mb2 looks for a directory named .mb2, where it stores some of its state. It is created implicitly by mb2 ... build and you can also create it explicitly with mb2 -t $VENDOR-$DEVICE-$PORT_ARCH build-init.

9 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:
â€¢ fastboot: Used by most Nexus devices
â€¢ 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:
â€¢ See any framebuffer console
â€¢ See any error messages of any kind during bootup
â€¢ Get any information relayed from your startup process
â€¢ 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 Androidâ€™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 doesnâ€™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
Ë“â†’udhcpd
```

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:
â€¢ If it immediately reboots (and especially if it later boots to recovery mode), SELinux is enabled, and all ports based on Android 4.4 (hybris-11.0) up to Android 9.0 (hybris-16.0) 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)

â€¢ 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
```

â€¢ 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. Itâ€™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 Sailfish OS port, a boot image with Sailfish OS-specific scripts will be built automatically. These boot im-
ages are then available as hybris-boot.img (for booting into Sailfish OS) and hybris-recovery.img (for debugging
via telnet and test-booting).

10 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

â€¢ Android Recovery flashed to your device
â€¢ The stock firmware image (for your version and device)
â€¢ The Android base release (for your version and device)
â€¢ 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 base release version you are going
to flash (which in turn is dictated by the Sailfish OS image you are going to flash).
While the Android base (e.g. CyanogenMod) .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
Android base .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 the Android
base image, and finally the Sailfish OS update. Please also note that you canâ€™t just take the latest stock ROM
and/or Android base ROM - both versions have to match the Android version against which the Sailfish OS
adaptation was built.

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

11 MANUAL INSTALLATION AND MAINTENANCE

This assumes you are booted into the Android base 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 (thereâ€™s an adbd running).

11.1 Extracting the rootfs via adb

Replace sailfishos-devel-hammerhead.tar.bz2 with the name of your rootfs tarball:

```
PLATFORM_SDK $
adb push sailfishos-devel-hammerhead.tar.bz2 /sdcard/
adb shell
su
mkdir -p /data/.stowaways/sailfishos
tar --numeric-owner -xvf /sdcard/sailfishos-devel-hammerhead.tar.bz2 \
-C /data/.stowaways/sailfishos
```

11.2 Flashing the boot image via adb

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

```
PLATFORM_SDK $
cd $ANDROID_ROOT
adb push out/target/product/hammerhead/hybris-boot.img /sdcard/
adb shell
su
dd if=/sdcard/hybris-boot.img of=/dev/block/mmcblk0p19
```

11.3 Flashing or booting the boot image via fastboot

```
PLATFORM_SDK $
cd $ANDROID_ROOT
# to smoke test a boot image without flashing it:
fastboot boot out/target/product/$DEVICE/hybris-boot.img
# to permanently flash an image to boot partition:
fastboot flash boot out/target/product/$DEVICE/hybris-boot.img
adb shell

su
dd if=/sdcard/hybris-boot.img of=/dev/block/mmcblk0p19
```

11.4 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
Androidâ€™s WLAN connectivity to connect to the Internet and download updates:

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

12 OTA (OVER-THE-AIR) UPDATES

You can setup to upgrade a Sailfish OS device over the air, a.k.a. OTA update.

12.1 Prepare the infrastructure

â€¢ Ensure your Sailfish OS version is at least 3.2.1 (3.4.0 for aarch64)
â€¢ Create file 20-mydomain.ini (rename â€œmydomainâ€ as you see fit) under $ANDROID_ROOT/hybris/droid-configs/sparse/usr/share/ssu/repos.d/ with the following content:

```
[release]
adaptation=https://mydomain.net/%(release)/%(vendor)-%(adaptation)/%(arch)/
```

â€¢ Substitute https://mydomain.net/ with your Web server address (including subpath if exists)
â€¢ The %(release)/%(vendor)-%(adaptation)/%(arch)/ format is advised, because itâ€™s the most future-proof. E.g. for the Nexus 5 this string would resolve to 4.5.0.19/lge-hammerhead/aarch64/
â€¢ Commit the above change to droid-configs (including updating the submodule, which introduces times-
tamped versioning, so updates get picked up)
â€¢ Make new image and ensure devices are flashed which will be receiving future updates
â€¢ Make some changes to your adaptation (e.g. fix some HW issue) and rebuild the affected part via
build_packages.sh, so that version numbers increase

12.2 Test for any breakages

Before deploying any updates to production, they must be tested first.

Prerequisites:
â€¢ Web server (e.g. Apache) running on HOST and accessible within network
â€¢ Directory listing doesnâ€™t need to be enabled
â€¢ Assuming Web serverâ€™s rootdir is /srv/http
Perform the following:

```
HOST $
. ~/.hadk.env
rm -rf /srv/http/sailfish-tmp-test-repo
cp -ar $ANDROID_ROOT/droid-local-repo/$DEVICE /srv/http/sailfish-tmp-test-repo
rm -rf /srv/http/sailfish-tmp-test-repo/repo
createrepo_c /srv/http/sailfish-tmp-test-repo
```

SSH into your device and execute (substituting https://mydomain.net with the address to your Web server):

```
DEVICE $
ssu ar sfos-test https://mydomain.net/sailfish-tmp-test-repo
devel-su -p pkcon install zypper
devel-su zypper refresh sfos-test
devel-su zypper dup --from sfos-test
```

Check that all the packages you touched are to be updated or removed as expected. Afterwards you can press
â€œYesâ€ to execute the update and check if the device functions as desired, also after reboot.

Once happy, clean up the testing environment:

```
DEVICE $
ssu rr sfos-test
HOST $
rm -rf /srv/http/sailfish-tmp-test-repo
```

12.3 Release into production for all users

Once successfully tested, deploy the stable packages to the release repo:

```
HOST $
. ~/.hadk.env
rm -rf /srv/http/$RELEASE/$VENDOR-$DEVICE/$PORT_ARCH
mkdir -p /srv/http/$RELEASE/$VENDOR-$DEVICE
cp -ar $ANDROID_ROOT/droid-local-repo/$DEVICE \
/srv/http/$RELEASE/$VENDOR-$DEVICE/$PORT_ARCH
rm -rf /srv/http/$RELEASE/$VENDOR-$DEVICE/$PORT_ARCH/repo
createrepo_c /srv/http/$RELEASE/$VENDOR-$DEVICE/$PORT_ARCH
```

To receive the update, each device will have to execute devel-su -p version --dup, and reboot when instructed.

12.4 Adding custom RPM packages

You can add any other RPM binary packages to the local build repository (i.e. packages that were not created by
running build_packages.sh). For example:

```
PLATFORM_SDK $
cd $ANDROID_ROOT
# Alternatively you can use `mb2 --output-dir ... build` instead of copying
cp -a path/to/custom-built.rpm droid-local-repo/$DEVICE
```

To make the devices of your users pull this RPM package in, ensure some other package or pattern requires it,
then test and deploy your repo as per instructions above.

12.5 Updating to the next Sailfish OS release

If another official Sailfish OS update has been released since you last published your HW adaptation update,
perform the following:

Update your SDK target device build environment (see how in the last paragraph of Setup the Platform SDK).
Alternatively, you can remove it and create a new one as per Installing Build Tools for Your Device.

Remove or backup your local build repository:

```
PLATFORM_SDK $
cd $ANDROID_ROOT
PREV_RELEASE=4.4.0.68
# adjust to the previous release version you were on
mv droid-local-repo/$DEVICE droid-local-repo/$DEVICE-$PREV_RELEASE
mkdir droid-local-repo/$DEVICE
```

Then rebuild all packages and a new image by executing build_packages.sh.

Afterwards test the rebuilt repo. The actual testing sequence on the device will be different:

```
DEVICE $
ssu ar sfos-test https://mydomain.net/sailfish-tmp-test-repo
ssu dr adaptation0
ssu re 4.5.0.19
# adjust to the actual version
devel-su -p version --dup
ssu rr sfos-test
ssu er adaptation0
```

Then reboot as and test device functionality.
Once satisfactory, publish your repo for all users.
Finally, to receive the update, each device will have to execute:

```
DEVICE $
ssu re 4.5.0.19
# adjust to the actual version
devel-su -p version --dup
```

NOTE: The %(release) in your self-hosted repo (visible via ssu lr) will get updated automatically after ssu
re.
After devel-su -p version --dup has finished, reboot as instructed.

13 MODIFICATIONS AND PATCHES

Running Sailfish OS on top of a Mer Hybris adaptation requires a few modifications to the underlying Android
base. We maintain forks of some repos with those patches applied.

13.1 Hybris Modifications to an Android base

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

13.1.1 Droid System

In order to work with libhybris, some parts of the lower levels of Android need to be modified:
â€¢ bionic/
â€¢ Pass errno from bionic to libhybris (libdsyscalls.so)
â€¢ Rename /dev/log/ to /dev/alog/
â€¢ TLS slots need to be re-assigned to not conflict with glibc
â€¢ Support for HYBRIS_LD_LIBRARY_PATH in the linker
â€¢ Add /usr/libexec/droid-hybris/system/lib to the linker search path
â€¢ external/busybox/: Busybox is used in the normal and recovery boot images. We need some
tional features like mdev and udhcpd
addi-
â€¢ system/core/
â€¢ Make cutils and logcat aware of the new log location (/dev/alog/)
â€¢ Add /usr/libexec/droid-hybris/lib-dev-alog/ to the LD_LIBRARY_PATH
â€¢ 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
â€¢ Remove various init and init.rc settings and operations that are handled by systemd
and/or Hybris on a Sailfish OS system
â€¢ frameworks/base/: Only build servicemanager, bootanimation and androidfw to make the
minimal Droid HAL build smaller (no Java content)
â€¢ libcore/: Donâ€™t include JavaLibrary.mk, as Java wonâ€™t be available

All these modifications have already been done in the mer-hybris GitHub organisation of forks from various
Android sources. If its android manifest is used, these patches will be included automatically.

In addition to these generic modifications, for some devices and SoCs we also maintain a set of patches to fix
issues with drivers that only happen in Sailfish OS, for example:
â€¢ hardware/samsung/: SEC hwcomposer: Avoid segfault if registerProcs was never called

13.1.2 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.
â€¢ Required Configuration Options: See $ANDROID_ROOT/hybris/hybris-boot/init-script
function check_kernel_config() for a list of required kernel options
â€¢ 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.

13.2 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.

14 DETAILED SUBSYSTEM ADAPTATION GUIDES

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 Sailfish OS linux
and Android, and lastly special features of Sailfish OS.

14.1 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 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 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:
â€¢ Enable CONFIG_INPUT_FF_MEMLESS kernel config option
â€¢ Disable CONFIG_ANDROID_TIMED_OUTPUT kernel config option
â€¢ Change maximum amount of ffmemless effects to 64 by patching ff-memless.c:
â€¢ 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
```

â€¢ 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+++ 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
```

â€¢ 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.
â€¢ Convert the android timed output vibra driver to support to ffmemless

- add â€œ#include <linux/input.h>â€
  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
â€¢ Non-Graphical Feedback Daemon (NGFD)

14.2 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.

The GStreamer-droid bridge is part of the integral build process. If you need to modify its source code, then
rebuild it via:

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

14.3 Camera

Launch the Camera app.
If if shows black screen and becomes non-responsive, enable the
audiosystem-passthrough-dummy-af package in the patterns and rebuild droid-configs.

If you find some parameters (such as ISO speed or other 3A settings) are missing from camera app, then itâ€™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.

14.4 Cellular modem

â€¢ Ensure Androidâ€™s RIL running ps ax | grep rild (expect one or two /system/bin/rild)
â€¢ If RIL is not running, check why it is not launched from /init*.rc scripts
â€¢ If itâ€™s launched, check where it fails with /usr/libexec/droid-hybris/system/bin/logcat -b radio
â€¢ 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!

â€¢ If itâ€™s firmware loading problem, trace all needed daemons in CM and their loading order as well as all
mounted firmware, modem, and baseband partitions.
â€¢ Once RIL is happy, then ofono can be launched. Unmask it if it was previously masked due to causing
reboots in Bootloops.
â€¢ If you still get no signal indicator in UI, remove SIM PIN and retry
â€¢ Also install ofono-tests package and run /usr/lib/ofono/test/list-modems

â€¢ Try to recompile latest ofono master branch from https://github.com/sailfishos/ofono
â€¢ If everything else fails, then stop and strace a failing daemon (either RIL or ofono) from command line
manually

14.4.1 Phone calls donâ€™t work (but SMS and mobile data works)

If the calling parties cannot hear one another, then the audiosystem-passthrough-dummy-af middleware
package is required, which should be enabled in the patterns.

14.5 Bluetooth

For bluetooth Sailfish OS uses BlueZ stack from linux.
TODO: bluetooth adaptation guide.
TODO: add detail about audio routing.

14.6 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.

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
Sailfish) accesses directly the WLAN driver bypassing wpa_supplicant.

The version of currently used wpa_supplicant can be checked from here:
https://github.com/sailfishos/wpa_supplicant

The version of used connman can be checked from here:
https://github.com/sailfishos/connman

14.6.1 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://github.com/sailfishos/dsme/commit/c377c349079b470db38ba6394121b6d899004963

14.7 NFC

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

14.8 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.

14.9 Audio

For audio, 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.

14.10 Sensors

Sailfish OS sensor support is based upon Sensor Framework at: https://github.com/sailfishos/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://github.com/sailfishos/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://github.com/sailfishos/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/Sensors.conf
which is supplied with the sensorfw backend plugin in QtSensors and a copy of it is already among your default
configs.

For Sailfish Core 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.

14.11 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 APIâ€™s for platform applications is exposed via nemo-keepalive package. See more details here:
https://github.com/sailfishos/nemo-keepalive

14.12 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
```

â€¢ NOTE 1: Please note that watchdog driver should disable itself during suspend.
â€¢ 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.

14.13 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
```

â€¢ LIPSTICK_OPTIONS should have â€œ-plugin evdevtouch:/dev/touchscreenâ€

14.13.1 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

15 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.

15.1 MCE libhybris Plugin

TODO

15.2 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 its 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

15.3 Non-Graphical Feedback Daemon (NGFD)

The Non-Graphical Feedback Daemon <https://github.com/sailfishos/ngfd> 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.
From here on shortened to NGFD.

TODO: add more detail about configuring NGFD.

15.3.1 Configuring Haptics

Sailfish OS uses NGFD to provide haptic feedback. We use a QtFeedback plugin to bridge it with NGFD. The
NGFD plugin is for providing feedback for events and alarms, it interfaces directly with QtFeedback that can be
used by 3rd-party applications.

When configuring haptics it is important to know if your device uses ffmemless or the LED/Droid based vibrator
interface.

To determine if your device uses the LED/native interface check for
/sys/class/timed_output/vibrator/enable or
/sys/class/leds/vibrator/activate. The exact path for these might be a little different in some cases, e.g. instead
of vibrator the path could contain foobar, foobar being the device name in this case. Check for down below Non-Graphic Feedback Daemon Native Vibrator Plugin for more.

If these files are not present it is very likely that your device uses ffmemless to control haptics. To verify if your
device uses ffmemless install the mce-tools package and run evdev_trace -i. If the listing contains a device with
the type EV_FF then your device uses ffmemless.

The qt5-feedback-haptics-ffmemless used before Sailfish OS 4.3 is deprecated in favor of the before mentioned
QtFeedback plugin.

When migrating away from qt5-feedback-haptics-ffmemless
/usr/lib/qt5/plugins/feedback/ffmemless.ini can be
removed without further intervention.

You can copy the Configuration file of the specific plugin used by your device to tune it fit better to your device.
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.

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.

15.3.2 NGFD PulseAudio Plugin

TODO

15.3.3 NGFD ffmemless Plugin

This is the main plugin handling vibra feedback for Sailfish OS for devices that use the ffmemless interface.
The default configuration file can be found in
/usr/share/ngfd/plugins.d/50-ffmemless.ini

https://github.com/sailfishos/ngfd/blob/master/data/plugins.d/50-ffmemless.ini.
The default configuration files can be over-ridden with setting environment variable:

NGF_FFMEMLESS_SETTINGS.

To set the environment variables add environment config file to your config package that installs to. Replace with
your <device> with the name of your device. E.g. mako, hammerhead etc.
/var/lib/environment/nemo/60-<device>-vibra.conf

And that file should look like below:

```
NGF_FFMEMLESS_SETTINGS=/usr/share/ngfd/plugins.d/ngf-vibra-<device>.ini
```

Now you can use the file to tune force feedback effects suitable specifically for your device.

For template to start making your own configuration files, just copy-paste the ngfd 50-ffmemless.ini <https://github.com/sailfishos/ngfd/blob/master/data/plugins.d/50-ffmemless.ini> default config files as the de-
vice specific files and then edit only needed bits.

An alternative instead of using the environment variable is duplicating the 50-ffmemless.ini in the same folder
with a different name such as 51-ffmemless.ini, NGFD will now pickup your configuration file instead of the stock
configuration file.

15.3.4 Non-Graphic Feedback Daemon Native Vibrator Plugin

This plugin uses the native kernel interface from the timed output driver or the led vibrator interface. The native
plugin doesnâ€™t require any configuration normally.

It is possible to set the path of the activation and duration controls as shown below if the plugin canâ€™t find these on
its own:

```
[droid-vibrator]
native.path
= /sys/class/leds/<device>/duration
native.activate_path = /sys/class/leds/<device>/activate
```

Replace <device> with the name of device directory for your vibration device.

It is the preferred method if the ffmemless plugin isnâ€™t used.

15.3.5 NGFD 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

15.4 PulseAudio Droid Modules

TODO - more information about how PA works

15.5 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:

â€¢ EGL_PLATFORM: For the Wayland Compositor, this needs to be set to fbdev on devices with older hw-
composer versions, and to hwcomposer for hwcomposer version 1.1 and newer. For best results, first try
fbdev, and if it doesnâ€™t work, try hwcomposer instead. For the Wayland Clients, this always needs to be
set to wayland.
â€¢ 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:
â€¢ 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, doesnâ€™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)
â€¢ 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)
â€¢ 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 libhybrisâ€™ 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 sup-
ported, 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.

15.6 SensorFW Qt 5 / libhybris Plugin

TODO

15.7 Build HA Middleware Packages

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

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 portâ€™s needs.

Follow the exact same compilation approach as with above packages. Known packages are:
â€¢ https://github.com/mer-hybris/unblank-restart-sensors - needed only by mako

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.

17 PACKAGE NAMING POLICY

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

17.1 List of naming rules

Packages that are arch-specific (e.g. aarch64), device-specific and contain $DEVICE in their name:

â€¢ 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)

â€¢ The package containing kickstart files for mic should be named **ssu-kickstarts-$DEVICE** (e.g. ssu-kickstarts-mako)

> PLEASE see the pdf for 17.1 and co!

18 HARDWARE ADAPTATION CHECKLIST

Before publishing the adaptation, at least the following features should be checked.
â€¢ Thermal sensor configuration for dsme
â€“ Even if we do not enforce any limits, CSD1 gets temperature info from dsme
â€“ Quick test:
dbus-send --system --print-reply --dest=com.nokia.thermalmanager \
/com/nokia/thermalmanager com.nokia.thermalmanager.battery_temperature
â€¢ memnotify patch to kernel + config for mce
â€“ Memory pressure normal|warning|critical affects for example browser
â€“ Quick test:
ls /etc/mce/*memnot*
â€¢ Watchdog driver in kernel + verify it works with dsme
â€“ We want the device to reboot if userspace gets hopelessly stuck
â€“ Some android kernels use hardware watchdog for kernel stuck detection
â€“ Quick test:
journalctl -b | grep 'dsme.*watchdog'
â€¢ usb-moded works
â€“ Detects charger and PC correctly
â€¢ USB diag mode works (optional)
â€“ Only needed for factory releases, and not even always for those
â€¢ USB gadget driver in kernel + verify it works with buteo-mtp
â€“ Android has some MTP logic imlemented at kernel and thus some FFS stuff we need is typically missing
â€¢ ssu config files
â€“ Verify ssu & ssu-sysinfo agree on results
â€¢ Vibra driver in kernel

â€“ Patterns choose android vibra, LED vibra or ff-memless (memoryless force-feedback devices)
â€“ ff-memless needs adding kernel driver
â€¢ Suspend works

â€“ If the device does not suspend, standby time will drop considerably
â€“ There is a CSD test for this (Hardware tests->All tests->System state)
â€¢ Resume via iphb works
â€“ Only â€œofficialâ€ way we have for scheduled wakeups from suspend
â€¢ Volume key probing & policy works
â€“ Display off -> no ringing volume change should happen
â€“ Display off -> audio playback volume should change
â€“ Both vol keys down -> UI snapshot should happen
â€¢ Power key works
â€“ Long press power key menu
â€“ Double presses
â€“ Loooong press shutdown in dsme
â€“ False double press reporting from a single press
â€¢ Proximity sensor works in suspend
â€“ We have built in assumption of having up-to-date p-sensor state
â€“ NB: If device does not have PS -> that must be configured
â€¢ Ambient light sensor works
â€“ Long sensor power up time -> can break display power on brightness
â€“ Kernel side filtering / odd delta reporting -> breaks auto adjustments
â€“ Total darkness should report â€œzero luxâ€
â€¢ LED works
â€“ Check the accuracy of colours and brightness
â€“ Blocking at sysfs write can make mce unresponsive
â€“ All but RGB LEDs probably require custom pattern config
â€¢ Proximity blanking during active call works
â€“ Some ports have weird problems here
â€¢ CSD config
â€“ HW features
â€“ Factory test set
â€“ Run-in test set
â€“ Masked/blacklisted tests
â€¢ abootsettings etc. when applicable
â€¢ Double tap works
â€“ There has been many devices where gestures are supported but touch driver uses odd concepts
â€¢ zram in kernel
â€¢ Look out for suspicious logging during bootup / shutdown
â€“ Faster/slower/just different -> odd things can/will happen
â€¢ usb-moded vs Android USB stuff in /*.rc
â€“ Device serial number is assumed to come from Android side logic

â€¢ Touch reporting
â€“ Seems many Android kernels have issues around display power cycling & finger on screen

â€¢ Act dead mode
â€“ What Android services are needed varies from one device to another
â€“ Act dead alarms need to be verified too
â€¢ Extra filesystems enabled in kernel where possible
â€“ BTRFS, F2FS, UDF, NFS, CIFS etc.

1 You can start the CSD tool either via command line (csd) or via Settings app: Go to â€œAbout Productâ€ and tap five times on the Build entry

19 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 â€œAS-ISâ€ 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 (â€œCCPLâ€ OR â€œLICENSEâ€). 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. â€œAdaptationâ€ 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 (â€œsynchingâ€) will be considered an
Adaptation for the purpose of this License.
b. â€œCollectionâ€ 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 Sec-
tion 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 assem-
bled 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. â€œDistributeâ€ 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. â€œLicense Elementsâ€ means the following high-level license attributes as selected by Licensor and indicated
in the title of this License: Attribution, Noncommercial, ShareAlike.
e. â€œLicensorâ€ means the individual, individuals, entity or entities that offer(s) the Work under the terms of this
License.

f. â€œOriginal Authorâ€ 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. â€œWorkâ€ 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. â€œYouâ€ 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. â€œPublicly Performâ€ 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. â€œReproduceâ€ 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 â€œThe original work was translated from
English to Spanish,â€ or a modification could indicate â€œThe original work has been modified.â€
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) (â€œApplicable Licenseâ€). 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 (â€œAttribution Partiesâ€) in Licensorâ€™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., â€œFrench translation of the Work by Original Author,â€ o
â€œScreenplay based on original Work by Original Authorâ€). 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 Authorâ€™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 Authorâ€™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 termi-
nology 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 â€œCreative Commonsâ€
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 Commonsâ€™ 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/.

HADK - Hot

Unofficial hadk-hot
Spend few minutes reading it all even briefly or if you are too lazy then waste much more time later fighting with known issues
These might be contenders to merge to: https://github.com/mer-hybris/hadk-faq
Improve and make PR to hadk-faq if you think that particular instruction is and will be useful for longer time.
The intention is to begin making this a wiki.
---------- ---------- -----

2026.03.27

*mal PSA: anyone with devices using android base 8 or newer make sure you update the libhybris used in hybris-hal build because latest qt5-qpa-hwcomposer-plugin requires updating that

* mal PSA: anyone building 5.0 adaptations in testing projects on OBS please add sailfishos_5.0 target to your adaptation project, I switched the repos to use major.minor versioning to remove the need to have separate target for each small release.

An example https://build.sailfishos.org/project/show/nemo:testing:hw:fairphone:fp5. So next time someone updates their devices from testing project the adaptation and common repo will change to use that 5.0 target

* "/bin/sh: ccache: command not found" when building pulseaudio-modules-droid from middleware.
Do:

```
sb2 -t $VENDOR-$DEVICE-$PORT_ARCH -R -m sdk-install zypper in ccache
```

* File /etc/bluez5/bluetooth/main.conf from droid-config-*-bluez5 conflicts with file from package bluez5-configs-mer.
Do:

```
sb2 -t $VENDOR-$DEVICE-$PORT_ARCH -m sdk-install -R zypper rm bluez5-configs-mer
```

* For some older sailfishos versions < 5.0 (or maybe < 4.6) 'zypper ref' doesn't really check whether url is correct or even reachable. It doesn't even mention failed refresh metadata for unavailable repositories. It seems to be related to ssu repository plugin.

2026.09.05: Android base specific fixes

Common

* since hybris-18:
- Do not use fetch-submodules when repo cloning repostories

* Since hybris-17:

- Add appropriate android_version_major: https://github.com/mer-hybris/hadk-faq/#hybris-17-1
- 'library "libandroidicu.so" not found': If you device do not use /odm then create sparse/odm/lib64/ (or sparse/odm/lib for 32bit minimediaservice) and put libandroidicu.so symlink pointing to /apex/com.android.runtime/lib64/libandroidicu.so or some similar path with that library. Usually adding only one simlink is enough, sometimes you may need also libicui18n.so and libicuuc.so. If your device use /odm and it contains valid data only in /odm/etc/ directory then mount odm partition to /.odm_root, create sparse/odm/lib* directory with needed symlinks in droid-config and bind mount /.odm_root/etc/ to /odm/etc/.
* Since hybris-16:

Before building hybris-hal run the following commands:

```
cd $ANDROID_ROOT/external
git clone --recurse-submodules https://github.com/mer-hybris/libhybris.git
cd $ANDROID_ROOT
hybris-patches/apply-patches.sh --mb
```

* Since android 8: some partitions can be mounted very early based on entries from devicetree instead android's fstab. Make sure you have all these partitions mounted in sailfish. For example samsung exynos7420.dtsi devicetree have defined efs partition which may be required to get wifi working correctly:

https://github.com/Exynos7420/android_kernel_samsung_exynos7420/blob/accf474903c0753fc1fcc4d25444481b701e7d94/arch/arm64/boot/dts/exynos7420.dtsi#L100

It can be checked also on device in:
    /proc/device-tree/firmware/android/fstab/

* For android 8-10 devices which use system-as-root without separate recovery you need to disable skip_initramfs handling so kernel can use hybris-boot initramfs. In kernel init/initramfs.c history find 2 following commits and revert them: "call free_initrd() when skipping init" and "Add skip_initramfs command line option". Alternatively, you can force in source code do_skip_initramfs = 0.

* If you do not package '/system' partition like in Sailfish X AOSP build instructions then you must have actual files instead symlinks in sparse/etc/selinux. Possible error message: Servicemanager: add_service Permission denied.

hybris-20 WIP

* Make sure you use latest hybris-patches

* "logd: libprocessgroup: Loading /etc/task_profiles.json for [2209] failed", then copy task_profiles.json from /system/etc to /etc and also add symlink /etc/task_profiles which points to /system/etc/task_profiles

* build *libui_compat_layer in HABUILD*

hybris-18

* dmesg errors:

```
kernel: loop_set_status: loop0 () has still dirty pages (nrpages=
kernel: print_req_error: I/O error, dev loop1, sector 0
kernel: Buffer I/O error on dev loop1, logical block 0, async page read
apexd: Failed to LOOP_SET_BLOCK_SIZE: Try again
```

Leads to apexd-bootstrap failure, ueventd start and reboot to fastboot mode. Apply this:

https://github.com/mer-hybris/android_kernel_sony_msm/pull/95
* 'Buffer I/O error on dev loop2, logical block 0, async page read'
Try adding loop.max_part=7 to kernel cmdline to makes loop images partitionable

* Starting ueventd can be caused by failure of some important services with reboot_on_failure option like apexd. This should be fixed in hybris-patches to ease early debugging.

* For devices which use dynamic partitions without real block devices for system, vendor or product either follow aosp hadk way and package all dynamic partitions or use

https://github.com/sailfish-on-nabu/parse-android-dynparts to map them and then mount by some script or custom systemd mount units. List of dynamic partitions can be found in BoardConfig*.mk and android's fstab.
Example solution for dynamic partitions handled via systemd:

- script which makes dynamic partitions available for mounting: https://github.com/mlehtima/droid-config-fp4/blob/devel/sparse/usr/lib/systemd/system/dmsetup.service

- example mount unit: https://github.com/mlehtima/droid-config-fp4/blob/devel/sparse/usr/lib/systemd/system/vendor.mount
- Do not forget to enable dmsetup and these mount units
- To prevent droid-hal creating incorrect systemd mount units add all dynamic partitions to makefstab_skip_entires similar to: https://github.com/mer-hybris/droid-hal-sony-lena/blob/master/rpm/droid-hal-common.inc#L20
- Be aware that some devices may have for example different vendor from stock/lineage super partition which can results in something doesn't work like wlan driver on spacewar

https://github.com/SailfishOSSpacewar/droid-system-Spacewar
- If your kernel have enabled selinux in permissive instead enforcing mode, files (not symlinks) in /etc/selinux are correct and logcat shows:

```
Unknown class service_manager
Could not self register servicemanager
```

then you may need selinux stubs for hw,vnd and servicemanager: https://github.com/mer-hybris/droid-config-sony-nile/blob/master/sparse/usr/libexec/droid-hybris/system/etc/init/hwservicemanager.rc#L4
99-qts
- If apexd-bootstrap fails with non zero code beacause of not updatable/flattened apexes:

```
This device does not support updatable APEX. Exiting
```

try experimental fix. This issue often leads to boringssl test failure, ueventd start and reboot to fastboot mode

```
Mixing copies of libbinder
```

You may need additional patches. Explanation: https://github.com/sailfish-on-karatep/karatep-port/blob/main/docs/rca/vndservicemanager-libbinder.md

---

2026.10.31: Recommended revisions

Using latest (default) instead following may cause issues
5.2.0
- droid-hal-configs = 6a9f35e
- pulseaudio-modules-droid = 14.2.110
- pulseaudio-modules-droid-jb2q = 14.2.104
- droidmedia = 0.20260522.0
5.1.0.x
- droid-hal-configs = 41cd1d9
- droid-hal-device = be67bbc
- pulseaudio-modules-droid = 14.2.106
- droidmedia = 81db5a7

You can use git reset --hard <tag/commit> to change revision and then build_packages.sh -b <path to source> to build except for droid-hal/configs/version
For some software which where build previously with different version you may get strange errors during compilation. In that case git clean -xdf sources might help

---

2026.09.29: Release specific issues

* 5.1.0.x
- You may need to update platform sdk before target/tooling
- Clean pulseaudio-modules-droid source directory before building new version

- If you loose internet access, 'dig jolla.com' timeouts try adding systemd-resolved to inet group:
https://github.com/mer-hybris/droid-config-sony-nile/pull/151 or disable CONFIG_ANDROID_PARANOID_NETWORK in kernel
* 4.5.0.18
- End of bluez4 compatibility: lipstick-jolla-home-qt5 requires libKF5BluezQt.so.6. kf5bluezqt was updated and bluez4 patches were droped
* 4.4.0.64:
- 'tracker-extract-3 Failed to load seccomp rules' on some <= 3.4 kernels. Either add check for ECANCELED in https://github.com/sailfishos/tracker-miners/blob/master/rpm/0003-Prevent-tracker-extract-failing-when-seccomp-loading.patch#L26 or add support for Seccomp BPF to kernel: https://bugzilla.mozilla.org/show_bug.cgi?id=790923. generic_seccomp-bpf_3.0.x patch is outdated and cause random kernel panic.
* 4.3.0.12:
- remove /var/cache/ssu and run 'ssu ur' before OTA upgrade to be sure that adaptation-common repo is enabled during OTA and packages will be upgrade otherwise you will have issues like: hybris.c:53: __resolve_sym: Assertion `ptr != NULL' failed: because too old packages: https://forum.sailfishos.org/t/4-3-0-12-issue-with-droidmedia/8963
* 4.2.0.x:
- mapplauncherd[1497]: Connection: getMountNamespace: stat failed for pid 1497: No such file or directory
Mount Namespace /proc/<pid>/ns/mnt was added in kernel 3.8 and probably backported to some 3.4 kernels. If your kernel do not have it then git am 12 patches (since "vfs: Add setns support for the mount namespace") for 3.4 kernel from: https://github.com/aosp-mirror/kernel_common/commits/556ef7fe2377ad919f20f108c2b5b205a57bbfae. Not needed for mapplauncherd >= 4.2.2
- Everyone using old hal versions of sensorfw and geoclue hybris plugins (used on android base versions <= 7, but some might use even on newer android bases): before building new release images you need to update patterns to manually fix the package names (to geoclue-provider-hybris-hal and hybris-libsensorfw-qt5-hal) if you need hal versions
An example of the fix https://github.com/mlehtima/droid-config-fp2-sibon/commit/1f6720efc02219ab7eabc185c8d07b44ebcfd421
* 4.0.1.x:
- psmisc, bash and procps have been replaced by busybox. busybox pidof cause /etc/profile.d/developer-profile.sh failure on login. busybox bash/ash is limited and also cause some strange issues. busybox killall is not compatible with bluetooth-rfkill-event <= 1.0.
To install non busybox version of bash and psmisc tools add to patterns:
Requires: gnu-bash
Requires: psmisc-tools
- Migrate from patterns to meta packages:
https://github.com/mer-hybris/hadk-faq#migrate-patterns-to-meta-packages
After transition make sure you have either patterns-sailfish-device-tools or zypper and busybox-static in patterns-sailfish-device-configuration-$DEVICE.inc:
https://github.com/mer-hybris/droid-hal-configs/blob/692f6aa48c3226631ab83263e13ed4ed9336e19e/patterns/templates/patterns-sailfish-

device-configuration-%40DEVICE%40.inc#L10-L25
- Move your systemd services from /lib/systemd/system/ to /usr/lib/systemd/system/
- adaptation0 repo can't be refreshed on fresh platform sdk < 4.1.0.x:
    sudo ssu domain sailfish

2026.09.16: General info and various issues

* Read https://github.com/mer-hybris/hadk-faq#android-base-specific-fixes before you start building

* sailfishos-porters channel logs search: https://piggz.co.uk/sailfishos-porters-archive/index.php

* Cheat Sheet: https://docs.sailfishos.org/Reference/Sailfish_OS_Cheat_Sheet

* Latest sailfishos version can be found in forum. Make sure you build sailfish image for latest minor release not the one from thread title.
* Latest sdk target url usually points to not latest sailfish release so always use versioned target url when preparing sdk. Most likely target/tooling minor release is not important. Example for 4.5.0.18:
    sdk-manage target install $VENDOR-$DEVICE-$PORT_ARCH
https://releases.sailfishos.org/sdk/targets/Sailfish_OS-4.5.0.18-Sailfish_SDK_Target-$PORT_ARCH.tar.7z --tooling SailfishOS-4.5.0 --tooling-url
https://releases.sailfishos.org/sdk/targets/Sailfish_OS-4.5.0.18-Sailfish_SDK_Tooling-i486.tar.7z --no-snapshot
* HADK:
- use 'repo init -u https://github.com/mer-hybris/android.git -b hybris-X' instead 'repo init -u git://github.com/mer-hybris/android.git -b hybris-X'
- gstdroidcamsrc-0.conf created by mk-cam-conf is no longer needed
- local manifest: $DEVICE.xml
First you need to find your device specific repository for example in github.com/lineageos (or in some custom github if your device is not supported by lineageos) like:
https://github.com/LineageOS/android_device_sony_pdx214 and add appropriate entry to local manifest. Then cm/lineage.dependencies file will tell you what other device specific repositories you may need like android_device_sony_sm8350-common according to:
https://github.com/LineageOS/android_device_sony_pdx214/blob/lineage-20/lineage.dependencies and add it also. In common repository you may find another cm/lineage.dependencies file with kernel or some hardware repository: https://github.com/LineageOS/android_device_sony_sm8350-common/blob/lineage-20/lineage.dependencies. Usually you need to add all of device specific repositories to local manifest. Some android_hardware_* repos might be already in https://github.com/mer-hybris/android/tree/hybris-18.1 xmls.
- fixup-mountpoints converts your by-name path defined in android's fstab and .rc files (find them on device or in device repo) for example: /dev/block/platform/msm_sdcc.1/**by-name**/system to block device for example: /dev/mmcblk0p23 which is needed to create working systemd mount units: /usr/lib/systemd/system/*.mount. These mount units will take care of mounting your partitions before droid-hal-init starts. So to create correct map (entry in fixup-mountpoints) for your

device you need entries from your fstab (find it in $ANDROID_ROOT/device/$VENDOR/*/rootdir/ or directly on device) and ls -l */by-name/* (according to HADK) output from device for coressponding block device (e.g. /dev/sda24). Do not assume that your device use "block/bootdevice/by-name/" or "block/platform/msm_sdcc.1/by-name" path for all partitions, check fstab instead! Do not follow _a/_b syntax in fixup-mountpoints used for official devices like pdx213 unless you really have these suffixes in fstab.
Entries like: system /system ext4 ro,bar
indicates dynamic/super partition and then needs to be handled differently
Example for hammerhead:

- fstab entry: /dev/block/platform/msm_sdcc.1/by-name/userdata /data ext4 noatime,nosuid....
-output of 'ls -l /dev/block/platform/*/by-name/' or 'mount' from device: ../../../../mmcblk0p28
Then entry in fixup-mountpoints should looks like:

-e 's block/platform/msm_sdcc.1/by-name/userdata mmcblk0p28 ' \
* mer-kernel-check: do not enable OPTIONAL filesystems unless you plan to use them.

* Make sure that after making hybris-hal, you have fstab* and ueventd*rc in out/target/product/$DEVICE/root/: https://github.com/mer-hybris/droid-hal-device/blob/53e6b3018148e4cc2a73e816ae979af8c5d2bfaa/droid-hal-device.inc#L498
If they are not there and you are using old android base then 'make modules' (on hybris-18 use 'allmod') can help figuring out name of modules you need and build it like: 'make fstab.qcom ueventd.rc'. If there are no such modules or files are still not copied to out/ directory then do it manually. Read $ANDROID_ROOT/droid-hal-$DEVICE.log to confirm that mount units for /system and other partitions are created. If android's fstab do not include '/system' (or '/' in case of system-as-root), '/vendor' or '/boot' entry then you might need to add them manually.

* system-as-root: https://source.android.com/docs/core/architecture/partitions/system-as-root
Generally it is a system partition with init files (part of ramdisk) and with another /system directory and if mounted incorrectly strange looking symlinks like /system/etc -> /system/etc. This should be handled automatically by droid-hal scripts provided your android's fstab in out/ contains proper entry for system partition. In that case system partition is mounted to /system_root and /system_root/system is mounted to /system. Some devices which use system-as-root uses '/system' mountpoint for system partition entry in fstab instead '/' which needs to be changed to get correct systemd mount units generated: system_root.mount and system.mount.
* For some devices ueventd*rc file do not contains all needed rules. Common missing rule is /dev/ion but can be anything else. In that case, copy ueventd*rc somewhere from /vendor partition or /system_root as ueventd.$DEVICE.rc to appropriate place in out/ directory.
* Format /data partition before flashing sailfish to remove full disk encryption (FDE) or file based encryption (FBE) which will cause permission issues later (failed: Required key not available)
* If you kernel crash very early without any logs in console-ramoops after enabling CONFIG_VT, check whether you have breaking commit "tty: move tty_port workqueue to be a kthread" in drivers/tty/tty_buffer.c recent history and revert it. There are usually 2 or 3 more related commits with "tty_port_set_policy" changes in drivers/tty/ which needs to be reverted too . Mostly for some lineage-15/16 kernel sources but not always.
* TWRP is known to fail when unpacking sailfish rootfs. Check /tmp/recovery.log for the exact reason: tar OOM or lack of bunzip2. There are many possible solution like downgrading TWRP,

telnet to hybirs-boot without flashing whole image and unpacking image manually, copying static busybox from build environment and using it to unpack image or switching from bzip2 to gzip for sailfish image.

gzip instead bunzip: https://piggz.co.uk/sailfishos-porters-archive/index.php?log=2023-04-13.txt#line293. Do not forget to update local repo metadata.
* When creating local manifest add device specific repositories from device/*/*/{cm,lineage}.dependencies files
* Updating and building droidmedia:

```
git fetch, pull and checkout to specific revision
make droidmedia (in HABUILD)
build_packages.sh --gg
```

Be careful if you use not recommended version for particular release. These steps will build only droidmedia but not droidmedia-devel which is used during gst-droid build done on jolla's servers
* To access scratchbox target as root: sb2 -t $VENDOR-$DEVICE-$PORT_ARCH -m sdk-install -R
* To access scratchbox target for building manually (cmake, make,..): sb2 -t $VENDOR-$DEVICE-$PORT_ARCH -m sdk-build
* HABUILD 'sudo: account validation failure, is your account locked?'
sudo pwck -q --root "$PLATFORM_SDK_ROOT"/sdks/ubuntu
* Latest ubuntu sdk do not have cpio installed which is required to create initramfs
* build_packages.sh fails with: "usage: mb2 [global-opts] <command> [command-opts]": update platform sdk to >= 4.3.0.15
* journal: 'systemd: Failed to create /user.slice/user-100000.slice/user@100000.service/init.scope control group: Permission denied
systemd: Failed to allocate manager object: Permission denied
systemd: Failed to create cgroup /user.slice/user-100000.slice/user@100000.service/*: Permission denied'

Apply "Remove /dev/cg2_bpf cgroup device" patch: https://github.com/mer-hybris/hybris-patches/commit/54008fe9fc67c13eff6d3d48f78bb268af7883be.
If that doesn't help then add systemd.legacy_systemd_cgroup_controller=yes to kernel command line
* Sailfish browser use now gmp-droid instead gst-droid. If you used "dont-use-droid-convert" quirk or have some playback issues then you probably need:

https://github.com/sailfish-on-fxtecpro1/gmp-droid/commit/81c69798fc7c785e72aea613a8b2cc5415c5b7ee
Now gecko-camera-droid-plugin?
* build_packages.sh fails: 'File /etc/ofono/ril_subscription.conf
  from install of
    droid-config-$DEVICE-1* (local-$DEVICE-hal)
  conflicts with file from package
    ofono-configs-mer-* (@System)'
    or similar for 'File /etc/ofono/binder.conf' and ofono-configs-binder conflicts

Add https://github.com/mer-hybris/droid-config-sony-nile/blob/ef51664245038906283c52aac22c7958d9a17107/droid-config-common.inc#L14-L15 to

droid-config-$DEVICE.spec, build packages and then:

```
sb2 -t $VENDOR-$DEVICE-$PORT_ARCH -m sdk-install -R zypper --plus-repo $ANDROID_ROOT/droid-local-repo/$DEVICE in droid-config-$DEVICE
```

* droid-hal-version-$DEVICE.log: Problem: the to be installed droid-config-$DEVICE-1-202111031520.armv7hl requires '$DEVICE-bluez-configs', but this requirement cannot be provided:
  sb2 -t $VENDOR-$DEVICE-$PORT_ARCH -m sdk-install -R zypper --plus-repo $ANDROID_ROOT/droid-local-repo/$DEVICE in droid-config-$DEVICE
* logcat: 'E/qdoverlay(2142): getRotDevFd failed to open rotator device' or 'HWRotator::Open: open /dev/mdss_rotator failed err = 13 errstr = Permission denied':

user is no longer in system group: https://github.com/mer-hybris/droid-hal-device/commit/1dac6f1ff3b58215003af61ba52bd91c5b177219
Change *_rotator group to graphics in: device/$VENDOR/*/ueventd.*.rc:/dev/*_rotator 0664 system graphics
* logcat: 'QC-time-services: Daemon:genoff_init_config: RTC initilization failed'. Disable time_daemon or vendor.time_daemon in disabled_services.rc: https://github.com/mer-hybris/droid-config-sony-ganges-pie/blob/master/sparse/usr/libexec/droid-hybris/system/etc/init/disabled_services.rc. On some older hybris base time_daemon is defined in some init*rc file which is parsed before /usr/libexec/droid-hybris (see logs). In that case you must disable service in that init*rc file. Sometime there can be also vendor.time_daemon defined somewhere in /vendor.
* Touchscreen doesn't work: remove :/dev/input/eventX from droid-hal-device.conf to enable auto detection of touchscreen node. If that doesn't help use evdev_trace (from mce-tools package) or 'cat /dev/input/eventX' to figure out correct node. (`for f in /sys/class/input/event*/device/name ; do printf "%s: %s\n" $f $(cat $f); done` can give an idea which device is which)
* Black background in homescreen or black images in gallery:

Set QT_OPENGL_NO_BGRA=1 in 99-qtscenegraph.conf: https://github.com/mer-hybris/droid-config-sony-nile/blob/master/sparse/var/lib/environment/nemo/99-qtscenegraph.conf
* After upgrading target rpm fail: 'error: db4 error(-30971) from dbenv->open: DB_VERSION_MISMATCH: Database environment version mismatch
error: cannot open Packages index using db4 - (-30971)
error: cannot open Packages database in /var/lib/rpm':

```
sb2 -t $VENDOR-$DEVICE-$PORT_ARCH -m sdk-install -R rm -rf /var/lib/rpm/__db.00*
```

* If your kernel do no have NETFILTER_XT_MATCH_QTAGUID then add "!" in mer-kernel-check script to make it optional: https://github.com/mer-hybris/mer-kernel-check/blob/master/mer_verify_kernel_config#L221
* Do not upload tmp/ or documentation.list in droid-config to github. These are temporary files recreated every time droid-config is build and they make your commits more obscure
* ' max77843-fuelgauge: driver failed to report `status' property: -22' this kind of spam in dmesg is usually caused by broken drivers which reports property which they do not support. Suppresing error printing in power_supply_sysfs.c can lead to several issues so real bug must be fixed in particular driver. For example following driver pretends to support status property:

https://github.com/edp17/android_kernel_samsung_exynos5433/blob/a06e248eafb831411afbf3bc57f4a47d50a89fc3/drivers/battery/max77843_fuelgauge.c#L22
but clearly status is not handled neither in max77843_fg_get_property nor in max77843_fg_set_property. Removing POWER_SUPPLY_PROP_STATUS should fix issue for this particular driver but other drivers might require different fixes. Fixing one bug might reveal another:
'max77843-fuelgauge: driver failed to report `charge_full' property': It support setter but not getter so add 'return -ENODATA in appropriate get_property function'
'bq51221-charger: driver failed to report `current_now' property'
Be careful: these are just examples and you are dealing with Li-Ion related drivers.
* USB cable detection is handled by usb-moded. It expects online/present and type events from power_supply devices for proper functioning. For debugging issues use 'udevadm monitor -p' and -D argument in /var/lib/environment/usb-moded/usb-moded-args.conf.
Some devices doesn't send online/present udev events on usb connection/disconnection but still set this parameters: /sys/class/power_supply/*/{uevent,present,online,type}. In that case adding power_supply_changed(<usb power_supply pointer>) to driver might be enough to get everything working:
https://github.com/edp17/android_kernel_samsung_exynos5433/blob/4dc34f7772e8e48bc2ecdc75bbfc4c0e81c9b3cd/drivers/battery/sec_battery.c#L2808
Alternative way is tracking ext_con or android_usb devices: https://github.com/sailfishos/usb-moded/pull/12
* Some devices needs forcing the USB to start in peripheral mode otherwise there will be no sign of usb connection in dmesg:
https://review.lineageos.org/c/LineageOS/android_device_xiaomi_sm6150-common/+/291494/4
https://github.com/sailfishos-on-sake/hybris-boot/blob/fa1ec67d3e8cfa564f61f6c1a0f65caf7c64fbfa/init-script#L197-L199

* Plugins: which one you should use is really device and adaptation/stock android version specific
- ofono plugin:
ofono-ril-plugin <= android 7
ofono-ril-binder-plugin <= android 10
ofono-binder-plugin >= android 10: binder-list -d /dev/hwbinder | grep IRadio (for supported radio interfaces)
radioInterface can be set in /etc/ofono/binder.d/*.conf
You may you need configuration files in sparse/etc/ofono, different for each plugin. See sony droid-config repos https://github.com/mer-hybris?q=droid-config- to figure out correct configs for plugin.
- pulseaudio plugin:
pulseaudio-modules-droid-jb2q <= android 10
pulseaudio-modules-droid >= android 11
For debugging you may add one or more -v to pulseaudio command. Take a look at /usr/lib/systemd/user/pulseaudio.service and /etc/sysconfig/pulseaudio and make sure you run it as user.
audio calls: https://github.com/mer-hybris/audiosystem-passthrough/blob/master/README.md

If you device have only 32bit audio library (ls /vendor/lib*/hw/audio*.so) e.g. /vendor/lib/hw/audio.primary.universal7420.so but without 64bit version e.g. /vendor/lib64/hw/audio.primary.universal7420.so then hidl_compat wrapper may help you:

https://github.com/Halium/android_vendor_halium_hardware/tree/halium-10.0

Clone repository and make sure you use correct branch for your android adaptation then you can build it similar to hybris-hal so: make audio.hidl_compat.default
Next figure out which audio.primary.*.so file is loaded via strace and adapt following mount unit.

https://github.com/mlehtima/droid-config-fp4/blob/devel/sparse/usr/lib/systemd/system/vendor-lib64-hw-audio.primary.default.so.mount

Enable android.hardware.audio* service if it's disabled in disabled_services.rc.
https://docs.ubports.com/gl/latest/porting/configure_test_fix/Sound.html
- sensorfw plugin:
hybris-libsensorfw-qt5-hal <= android 7: test_sensors
hybris-libsensorfw-qt5: >=android 8: binder-list -d /dev/hwbinder | grep ISensors

- gbinder api level: /etc/gbinder.conf
If binder-list -d /dev/binder doesn't show interface then try changing api level
possible values:

https://github.com/mer-hybris/libgbinder/blob/2b2e22aced55d023f2419817e92935e78675c108/src/gbinder_config.c#L189

- bluetooth:
On recent android bases you need to install bluedroid and enable HCIVHCI and some other configs:

https://github.com/mer-hybris/android_kernel_sony_msm/blob/fbfc0728b6a708ed67791ccca23821777d6c89eb/arch/arm64/configs/aosp_lena_pdx213_defconfig#L920

- Vibrator

https://hadk.sailfishos.org/subsystems/#vibration-force-feedback
https://hadk.sailfishos.org/middleware/#non-graphical-feedback-daemon-ngfd
partially outdated informations: https://github.com/mer-hybris/hadk-faq#id54
There few plugins:
- ngfd: libngfd_ffmemless.so: have_ffmemless in droid-hal-version
- ngfd-plugin-native-vibrator: have_vibrator_native in droid-hal-version
- ngfd-plugin-droid-vibrator: have_vibrator in droid-hal-version
- ngfd-plugin-binder-vibrator: have_vibrator_binder in droid-hal-version
- libngf-qt5-qtfeedback: It bridges haptic feedback support from ngfd to qt applications. Deprecates qt5-feedback-haptics-*.
- qt5-feedback-haptics-droid-vibrator and qt5-feedback-haptics-native-vibrator: depracated
- qt5-feedback-haptics-ffmemless: depracated

Add appropriate have_* variable in your droid-hal-version

* suspend
- test with: mcetool --get-suspend-stats or systemdatascope app

- You may need to enable autosleep in kernel: CONFIG_PM_AUTOSLEEP=y
- If you usb mode doesn't change from peripheral/none when you plug in/out usb. Permanent peripheral mode might block suspend. Following service should help:

https://github.com/VerdandiTeam/usb-state-notifier

Make sure it uses correct '/sys/devices/platform/soc/4e00000.ssusb/mode' path
TO CONFIRM:
- Make sure you have this ne patch applied: "Add kernel wakelock to Android suspend":

https://github.com/mer-hybris/hybris-patches/pull/33

Then build patched service: 'make android.system.suspend-service'

and add rc service to start patched version of android suspend: https://github.com/sailfishos-sony-nagara/droid-config-sony-nagara/blob/main/sparse/usr/libexec/droid-hybris/system/etc/init/android.system.suspend-service.rc

* 'android.hardware.keymaster@3.0-impl: Configure failed'
Set correct OS version and patch level for your *boot.img

https://github.com/sailfishos-sony-tama/droid-hal-img-boot-sony-tama-pie/commit/4712b68c2f7387b5c342d4a3fe492d78a568bf50

https://wiki.postmarketos.org/wiki/Category:Bootloaders/Keymaster
* Random kernel <= 3.10 crash at:
'fs/sysfs/dir.c' called by sysfs_remove_dir: BUG_ON(sd->s_flags & SYSFS_FLAG_REMOVED);

Apply "kobject: grab an extra reference on kobject->sd to allow duplicate deletes" patch
* Some 3.4 and 3.10 kernels requires following patches to get systemd and sailjail/firejail working

- Revert "VFS: make vfs_fstat() use f[get|put]_light()

- vfs: make O_PATH file descriptors usable for fstat()

- vfs: allow O_PATH file descriptors for fstatfs() or or this one

- proc: show mnt_id in /proc/pid/fdinfo
or bunch of others

- https://github.com/shr-distribution/linux/commits/hammerhead/3.4/halium-9.0

- https://github.com/shr-distribution/linux/commits/mako/3.4/halium-9.0

- net loopback: Set loopback_dev to NULL when freed
Also make sure you have enabled CONFIG_.*_NS in kernel.
* ohmd failing twice or more with error: 'ohmd.service: Failed at step NAMESPACE spawning /usr/sbin/ohmd: No such file or director' is caused by mising namespace options from mer-kernel-check and CONFIG_USER_NS
* wpa3 support (allowed options: none, mixed, full): https://github.com/mlehtima/droid-config-fp5/blob/00b78f4d2df9e09980fe6cda0b6a1520c7406ff2/sparse/etc/connman/main.conf.d/10-fp5.conf#L5

* Updated promote.py for python3: https://pastebin.com/KC4LKydz. Make sure you use right "apiurl = https://build.sailfishos.org" in ~/.oscrc

* bluez4 was dropped: https://github.com/mer-hybris/droid-hal-configs/pull/245

* Alternative to lipstick-hack (aka surfaceflinger-hack):

https://wiki.merproject.org/wiki/Adaptations/faq-hadk#Alternative_to_lipstick-hack_.28aka_surfaceflinger-hack.29

* stracing android services

Because android services might be started by specific user/group then regular stracing may hide or show false culprit. Sometimes it may be good idea to strace service directly in *.rc file which it is defined:

```
-service tad /vendor/bin/tad /dev/block/bootdevice/by-name/TA 0,16
+service tad /usr/bin/strace -ff -o /data/strace/tad.strace /vendor/bin/tad /dev/block/bootdevice/by-name/TA 0,16
```

also create writable directory for generated files:

```
mkdir -m777 /data/strace/
```

2026.07.10: Generating logs:

* Always provide full logs with information of what you did so far, full command you are running with arguments and shell prompt in paste so it's clear what you execute and where: host, device, sdk, target sdk or ubusdk. If you are debugging issues on device: reboot (so it is not filled with too much noise) and make sure logs contains early boot messages. It's better to highlight relevant part than providing partial log. Do not paste long logs directly to channel, use paste services instead:

https://pastebin.com - full of ads, https://paste.opensuse.org - raw view links expire quickly or similar. ubuntu one without raw view and lack of horizontal scrollbar is worst choice. Better alternatives are welcome

* Increase your terminal buffer to not loose early logs
* Always upload your latest changes in droid-config/hal/kernel so it's easy to inspect them. Crazy ideas: read logs and search for errors from the top before you upload them
* In TWRP sailfish root path / is under /data/.stowaways/sailfishos/
* Connecting device to pc shortly after powering it on and running dmesg in your host system can reveal whether you are booting hybris initframfs, android init or there is no sign of life on usb from your device
* If ifconfig shows that inet address is not assigned and you can't telnet to device:

https://docs.halium.org/en/latest/porting/debug-build/early-init.html. Sometimes init*usb*.rc file can mess up your usb connection after droid-hal-init is started. To avoid it you must prevent that particular *.rc file from being included and parsed. This is probably done in some more general init*.rc file. Sometimes masking usb-moded can help also untill you get gui working.

* If your device bootloops but you can force boot straight to TWRP then cat /proc/last_kmsg or /sys/fs/pstore/console-ramoops might contains logs from faulty boot but only if kernel crashed. Make sure this log are really from sailfish boot attempt instead TWRP or android.
* If you are on 23 telnet port then device stuck in initramfs, cat init.log and diagnosis.log will tell you why switching to real rootfs failed

* If your device bootloop fast enough or reboot to some different mode so you can't access telnet check in TWRP whether /data/.stowaways/sailfishos/init.log exists if no then your device did not pass or even reach initramfs stage. In that case make sure your hybris-boot.img contains valid initramfs: 'abootimg -i hybris-boot.img'. If init.log is generated you can stop booting device in /init-debug script by creating /data/.stowaways/sailfishos/init_enter_debug2 which might allows you to telnet.
* If your device reboots after 60s without any sign of life in dmesg then check whether you have enabled CONFIG_USB_CONFIGFS_*RNDIS
* dmesg: usually useful to confirm whether selinux is disabled at boot time: 'SELinux: Disabled at boot' for hybris <=16 or initialized: 'SELinux: Initializing' for hybris >= 17. Also if something fails very early before systemd is started or some droid-hal-init logs are partialy in dmesg and partially in journal. Kernel buffer can be small so you need to be fast to grab early boot messages.
Arguments:
    -w (live log, not supported on very old kernels)
* journalctl: enable persistent logging if you have trouble with getting access to device, you kernel is noisy or you are too slow so early boot logs are lost: https://github.com/mer-hybris/hadk-faq#persistent-journalctl. Journal binary log can be also retrieved from /var/log/journal via TWRP.
Arguments:
    --no-pager (shows everything without paging)
    --no-tail -f (shows everything and continuously prints new messages)
    -b0 (only from current boot, useful if you have persisten logging enabled) -b-1 (from previous boot)
* /usr/libexec/droid-hybris/system/bin/logcat: for android part debugging. Can produce different logs as regular user than root user.
* strace <command>
Arguments:
   -f (trace child processes - if you are not sure just always add it)
   -e file (for tracing files)
   -s256 (increase max string size)
* If droid-hal-init do not print: 'Parsing file *.rc' or you have: 'output lines suppressed due to ratelimiting' in dmesg then add 'printk.devkmsg=on' to kernel cmdline to enable unlimited logging from user space applications
* Always add audit=0 to kernel bootparams to suppress audit spam and clean up logs unless you plan to use audit/selinux.
* binder-list from libgbinder-tools package:

```
binder-list
binder-list -d /dev/hwbinder
binder-list -d /dev/vndbinder
```

---------- ----------

No sign of life

If you flashing hybris-boot.img leads to no sign of life from device neither in host dmesg nor in device then make sure boot image is correctly created. Best way to do this is by comparing with lineage's boot.img via unpack_bootimg. If all looks sane then building hybris-boot without any defconfig changes is good idea. As a last resort recreate manually lineage boot.img if that shows same sign of life then try to replace lineage's ramdisk with hybris one. Make sure tarball uses same format.

TODO: vendor_boot boot ing issues

I have telnet but not gui:

* Check in .config, zcat /proc/config.gz or dmesg whether selinux is set up correctly. When you build kernel, defconfig is used as source from which out/target/product/$DEVICE/obj/KERNEL_OBJ/.config is generated. If you forgot about dependencies in defconfig then it will not match generated .config. Sometimes defconfig is stripped from some configs so it's good to regenerate it: build kernel without any hybris changes in defconfig, copy generated .config back to defconfig and commit that change. Disabling CONFIG_AUDIT which is part of CONFIG_SECURITY_SELINUX dependencies is common mistake.

* Search for "droid mount" in journal and make sure /system and other partitions are mounted. You should have similar partitions mounted like on android. android's fstab can give you a clue but some partitions are mounted differently

* Search for firmware loading failure. If particular driver needs firmware before partitions are mounted and droid-load-firmware.sh is not triggered then building driver as module and creating systemd service (like for wlan) which will modprobe it might help.

* droid-hal-init (modified android init) must not fail and usually can't be started twice unless it fails on early stage or you clean up some files it created before starting it again. If you want to strace it then mask service and reboot: 'systemctl mask droid-hal-init'. logcat can be used only after droid-hal-init started.

* compositor used by sailfish is called lipstick. For debugging gui issues use 'systemctl mask user@100000' and reboot device. This will prevent lipstick starting/failing and ease test_hwcomposer/minimer use.

To test android part, mask user@100000, reboot an start surfaceflinger:

```
/system/bin/surfaceflinger
# On android >= 11 you may also need:
ANDROID_ROOT="/system" /system/bin/bootanimation
```

2025.06.15: OBS (build.sailfishos.org) tips and tricks

Extends: https://github.com/mer-hybris/hadk-faq#obs-build-and-over-the-air-updates-ota

Useful OBS Notes: https://github.com/sailfishos-chum/main/blob/main/GettingStarted.md
* For account conntact Keto at #sailfishos* IRC channel or on sailfish forum with username and email

* OBS API url for osc: https://build.sailfishos.org
* Follow fp2-sibon example repos and adapt for your device:

https://build.sailfishos.org/project/show/nemo:devel:hw:fairphone:fp2-sibon and

https://build.sailfishos.org/project/show/nemo:testing:hw:fairphone:fp2-sibon
* You can add new relases using "Meta" tab instead via "Add repositories"
* '<path project="sailfishos:*" repository="latest_armv7hl"/>' should be latest in projects list in "Meta"
* devel repo is for you and your testers development. It should have only one release sailfish_latest_armv7hl. Testing repository might have releases for every no EA (Early Access) sailfish version like sailfishos_4.2.0.21.
* Copy and adapt "Project Config" from fp2-sibon to your device:

https://build.sailfishos.org/project/prjconf/nemo:testing:hw:fairphone:fp2-sibon
* OBS targets are not added at the same time as new sailfish os version is announced. Search for "sailfishos:4." in https://build.sailfishos.org/project to be sure that you can build for particular release. "sailfishos:latest" for devel repository is updated only after some time when full release is out. Confirm here: https://build.sailfishos.org/repositories/sailfishos:latest
* Always disable old releases before you create new in testing repo. You can do this in "Repositories" tab by setting "Build Flag" to disable or manually in "Meta" tab (see fp2-sibon example). If you do not do this then you will override your old release. Alternatively you can create subproject for each release which will ease maintenance
* Add droid-config/hal/version and other adaptation packages but do not add packages like qt5-feedback-haptics-native-vibrator or community-adaptation-* which are already in repositories enabled on your device: https://build.sailfishos.org/project/show/nemo:testing:hw:common, https://releases.jolla.com/releases/$RELEASE/jolla-hw/adaptation-common/$PORT_ARCH/ (since 4.3.0) or in https://releases.jolla.com/releases/$RELEASE/jolla/$PORT_ARCH/ unless you need to build some different version.
* Only droid-hal and droidmedia needs to by uploaded because they are build from android sources. For all other packages, create _service file and fill it with:

https://build.sailfishos.org/package/view_file/nemo:testing:hw:fairphone:fp2-sibon/gst-droid/_service?expand=1. You can set revision to HEAD but sometimes latest version for particular package which is under development can not be used then you can set revision to tag or git commit hash. OBS will fetch sources from git and build automatically. If you want to rebuild package then use "Trigger services" button.

* osc can be also used for uploading new packages: https://wiki.merproject.org/wiki/Osc_Setup
* Package name determine which spec file will be used to build it. Here sensorfw-qt5-hybris.spec will be used https://build.sailfishos.org/package/show/nemo:testing:hw:fairphone:fp2-sibon/sensorfw-qt5-hybris but if you use binder backend then you need sensorfw-qt5-binder.spec

thus: https://build.sailfishos.org/package/show/nemo:testing:hw:oneplus:cheeseburger/sensorfw-qt5-binder
* Package blocked for very long time with following message: "downloading 1 dod packages". Ping OBS maintainer
* Your $ANDROID_ROOT/*.ks file which is used by "mic" during image creation by default use local repository on your disk for adaptation packages and common/jolla repos for other packages. Also build_packages -i/--mic overrides your *.ks file. One way to use your new OBS repo is: download droid-config-$DEVICE-ssu-kickstarts*.rpm:

https://build.sailfishos.org/package/binaries/nemo:testing:hw:fairphone:fp2-sibon/droid-config-fp2-sibon?repository=sailfishos_4.2.0.21 and unpack *.ks with your new adaptation-community-$DEVICE repo, update droid-local-repo metadata then run mic manually:

```
EXTRA_NAME="extra"
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 #--verbose --debug
```

* .ks file change sometimes together with new target and droid-config so download new one for every release
* mic installs only one package patterns-sailfish-device-configuration-$DEVICE defined in *.ks file when sailfish image is created. That means every other package installed in image is pulled by droid-config/patterns/patterns-sailfish-device-configuration-$DEVICE.inc. For optional packages which can be removed later without uninstalling other adaptation packages use weak dependencies like: "Recommends: jolla-developer-mode" instead "Requires: jolla-developer-mode".
* Ports do not recieve updates in Settings page. For upgrade disable 3th party repos and use "ssu release $RELEASE; version --dup". If it fails for some unknown reason try zypper ref; zypper dup --details to figure out what is broken.
* When using OBS web interface to upload rpm package with '+' in file name OBS will rename plus sign to space and copy will fail. To solve this explicitly set correct "Filename" at "Add File" page when you upload such rpms.

* The OBS API url used to osc changed, it's now https://build.sailfishos.org . If you have local checkout and you want to replace to url in those use something like this:

```
grep --include=_apiurl -rl build.merproject.org . | xargs sed -i 's/build.merproject.org/build.sailfishos.org/g'
grep --include=_apiurl -rl api.merproject.org . | xargs sed -i 's/api.merproject.org/build.sailfishos.org/g'
```

depending on which url you had been using before. In addition to changing it in local checkouts also fix the url in .oscrc
* _pattern is no longer used for new releases, forget about it