# Halium Documentation

Halium Documentation Contributors

Nov 20, 2018

# Porting Guide

Welcome to the documentation for Halium, the collaborative project to unify the Hardware Abstraction Layer for projects which run GNU/Linux on mobile devices with pre-installed Android.

CHAPTER 1

# Contents

## 1.1 First steps

This page contains the first steps you should take to start a Halium port.

### 1.1.1 Getting help

If you get stuck at any point during the porting process, we're here to help! You can contact us via the following support channels:

- IRC: #halium on Freenode
- Matrix: #halium:disroot.org
- Telegram: @halium

When you contact us, please link to the section of documentation you're stuck on. In the HTML version, you can copy links by hovering your pointer over a heading, right-clicking the 'link' icon that appears to the right of it, and selecting "Copy link location" or its equivalent in your browser.

### 1.1.2 Pick an Android target device

If you're here, you probably already have a device in mind that you wish to port to. However, we still encourage you to port to devices that meet the following requirements:

- **Kernel:** Halium currently requires a device with a kernel greater than or equal to version 3.1.0 - older device kernels are not compatible with the glibc build in the root filesystem being used. Check your device's kernel version by finding "Kernel Version" in the About page of your Android settings.
- **RAM:** While 1GB of RAM is sufficient to start the OS, it is recommended to have greater than 2GB to have a good end user experience.
- **Chipset:** Try to avoid Mediatek chipsets, they are not open-sourced and so there is rarely a usable Android source tree available for them.

- **Storage:** 16GB of storage is generally enough for any Halium-based OS.

### 1.1.3 Collaborate

Head over to the list of ports and check if someone is already working on this device. If it is started, collaborate with those porters.

### 1.1.4 Document your target device

Head over to *How to contribute documentation* to learn about adding your device's information to our device overview. Adding your device will also lead you to looking at other Halium devices, some of which may have similar hardware to yours. You can use fixes made in similar devices to fix your own.

### 1.1.5 Set up your build device

Now you will need to install packages on the computer you wish to build Halium with.

#### Debian (Stretch or newer) / Ubuntu (16.04 or newer)

If you are on the `amd64` architecture (commonly referred to as 64 bit), enable the usage of the `i386` architecture:

```
sudo dpkg --add-architecture i386
```

Install the required dependencies:

```
sudo apt install git gnupg flex bison gperf build-essential \
  zip bzr curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
  libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
  libgl1-mesa-dev g++-multilib mingw-w64-i686-dev tofrodos \
  python-markdown libxml2-utils xsltproc zlib1g-dev:i386 schedtool \
  repo liblz4-tool bc lzop
```

#### Arch

If you have a pure `amd64` running you need to add `[multilib]` repository to your `/etc/pacman.conf` . This will allow you to install and run `i686` packages. Please refer to https://wiki.archlinux.org/index.php/multilib Assuming you have already installed `base-devel` group; if not, then please install `base-devel`.

Install the required dependencies from AUR:

```
git clone https://aur.archlinux.org/halium-devel.git && cd halium-devel && makepkg -i
```

**Note:** Arch uses python3 as default version for Python, which may cause some errors while building. Using a Python virtualenv2 is highly recommended. Please refer to https://wiki.archlinux.org/index.php/Python/Virtual_environment for instructions on setting the Virtual Environment.

**Todo:** Add information for installing packages on other distros

## 1.2 Get Halium source

Now that you have a target device selected and your build device ready, it's time to get the sources for your target together. Let's begin by initializing your source tree.

### 1.2.1 Initialize and download source tree

Make yourself a new directory to put your Halium source in:

```
mkdir halium && cd halium
```

This directory will be called BUILDDIR in the remaining part of the guide, when necessary, to avoid confusion.

If the target device has Android 7.1 or LineageOS 14.1 support, it's recommended to select `halium-7.1`:

```
repo init -u https://github.com/Halium/android -b halium-7.1 --depth=1
```

If your device does not have Android 7.1 or LineageOS 14.1 support but has support for Android 5.1 or CyanogenMod 12.1, select `halium-5.1`

```
repo init -u https://github.com/Halium/android -b halium-5.1 --depth=1
```

`halium-7.1` is based on LineageOS 14.1

`halium-5.1` is based on CyanogenMod 12.1

Now that you have initialized the Halium tree (a shallow copy due to –depth=1), you can sync all repositories included in it. This will probably take a while as it downloads several GBs:

```
repo sync -c -j 16
```

In case you want to get the full repo history at some point in the future because you want to patch, contribute, etc. You can execute the following command from the repository directory

```
git fetch --unshallow
```

### 1.2.2 Adding your device-specific source

Okay, so now you have the default manifest for Halium. This will enable you to download the basic Android sources used to build Halium, but you'll need to find device-specific files. These enable the build system to make Android for your device.

If there's any part for you to make a mistake on, this will be the one. Finding the files isn't hard, but it might take you a couple of tries to get repo's local manifest created correctly.

First, you'll want to find the repositories for your device on LineageOS's GitHub organization. You can do this by typing your device's codename into the search box. You'll want the device repository, `android_device_[manufacturer]_[device]`. Take down this name, as you'll need it later.

There will be a `cm.dependencies` or `lineage.dependencies` file in that repository that will tell you all of the other repositories that your device is reliant upon. Keep this file around as you will need it in a little bit.

Navigate into your Halium directory and create the file `halium/devices/manifests/[manufacturer]_[device].xml`.

Paste the following into the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>

</manifest>
```

We recommend cloning all of the repositories that you'll find in the next steps to a personal source archive (aka, fork or import them on GitHub), then adding those personal forks to the manifest. This will help you save and share your work later.

### The Device Repository

Next, we'll fill the manifest with information. Start with your device repository. Create the following line between the <manifest> and </manifest> tags, replacing the information inside the square brackets with your own:

```
<project path="device/[manufacturer]/[device]" name="[repository name]" remote=
↪"[remote]" revision="[revision]" />
```

**Note:** The `revision` attribute may be omitted if the default revision for this remote is the one that you wish to use.

If you're not sure of your remote, jump down to *Remotes*.

### Dependencies

Now create more lines like the previous, using the `cm.dependencies` or `lineage.dependencies` file you found earlier in your device repository. This file lists all of the other repositories that you need to build for your selected device. It's listed in a fairly straightforward way, so create a line for each of the entries in there using the following template:

```
<project path="[target_path]" name="[repository]" remote="[remote]" revision=
↪"[revision]" />
```

The target path is found in the repository's name. The preceding "android" or "proprietary" is omitted and underscores are replaced with slashes. For example, `android_device_lge_bullhead` goes in `device/lge/bullhead`.

### Vendor blobs

Vendor blobs go in the `vendor/` folder in your Halium source tree. You will need to find these in a repository or otherwise and add them to your source tree. Check if your device's vendor is listed in TheMuppets' GitHub organization, then open the repository to see if your device's codename is inside on your desired branch.

If you are unable to find your device in TheMuppets, you will need to find another repository with the vendor files somewhere.

### Remotes

A remote entry specifies the name, location (fetch) prefix, code review server, and default revision (branch/tag) for source.

You can create a remote by adding a `remote` tag to your manifest.

```
<remote  name="aosp"
     fetch="https://android.googlesource.com"
     review="android-review.googlesource.com"
     revision="refs/tags/android-7.1.1_r25" />
```

Only the name, fetch, and revision attributes are required. The review attribute specifies a Gerrit Code Review server, which probably won't be useful for initial porting purposes.

For example, let's say that you have a bunch of repositories at `https://github.com/MyUserName/` and your desired branch name is `cm-14.1` in each. You would create a remote as follows and place it into your local manifest:

```
<remote name="mun"
    fetch="https://github.com/MyUserName"
    revision="cm-14.1" />
```

There are also some remotes available to you by default, though they differ between halium-5.1 and 7.1. The following tables will help you identify these. See more information on these remotes by viewing the top of `.repo/manifest.xml` in your initialized BUILDDIR.

#### halium-7.1

These remotes are available to you by default in halium-7.1:

| Remote Name | Remote Description, URL |
|---|---|
| aosp | Android Open Source Project, https://android.googlesource.com |
| los | LineageOS, http://github.com/LineageOS |
| hal | Halium (link to GitHub root for legacy reasons), http://github.com |
| them | TheMuppets, http://github.com/TheMuppets |

If you do not specify a remote, `aosp` is assumed.

#### halium-5.1

| Remote Name | Remote Description, URL |
|---|---|
| phablet | Canonical Ubuntu Phone compatibility, https://code-review.phablet.ubuntu.com |
| aosp | Android Open Source Project, https://android.googlesource.com |
| cm | CyanogenMod, https://github.com/CyanogenMod |
| ubp | UBports (link to GitHub root for legacy reasons), https://github.com |
| halium | Halium (link to GitHub root for legacy reasons), https://github.com |
| ab2ut | Vendor blobs for UBports builds, https://github.com/ab2ut |

If you do not specify a remote, `phablet` is assumed.

### 1.2.3 Sync

Now that you've got your manifest filled out, simply type the following to get all of your source (replace DEVICE with your device's codename):

```
./halium/devices/setup DEVICE
```

This will first link your manifest from Halium devices to `.repo/local_manifests/device.xml` and then sync all repositories. This can take a while as it will download up to 2 GB of sources. If you have a fast connection, you may set an extra `JOBS=[number]` environment variable at the beginning of the command to make more parallel downloading jobs. We generally recommend 12, which is the default.

### 1.2.4 Document

After following these steps, create an issue on the Halium project management repository to document your porting progress. Also create a pull request containing your manifest on the Halium devices repository. You should link the manifest on Halium devices in your project management issue. Alternatively you can also use a link to the pull request, if the manifest was not merged already.

### 1.2.5 Next steps

Now that you've got your source tree downloaded, you can move on to the next page where we'll start to build it!

## 1.3 Build Halium

You've got all of your source downloaded, now it's time to start building!

### 1.3.1 Initialize

First we need to initialize the environment using the envsetup.sh tool. Enter BUILDDIR in a terminal and type:

```
source build/envsetup.sh
```

This will give you an output that looks like this:

```
including device/lge/bullhead/vendorsetup.sh
including vendor/cm/vendorsetup.sh
including sdk/bash_completion/adb.bash
including vendor/cm/bash_completion/git.bash
including vendor/cm/bash_completion/repo.bash
```

#### For Halium-5.1

We need to choose the target to build using the lunch command:

```
lunch
```

The output of this command will look something like this:

```
You're building on Linux

Lunch menu... pick a combo:
 1. aosp_arm64-eng   4. aosp_mips-eng     7. cm_bacon-eng
 2. aosp_arm-eng     5. aosp_x86_64-eng   8. cm_bacon-user
 3. aosp_mips64-eng  6. aosp_x86-eng      9. cm_bacon-userdebug

Which would you like? [aosp_arm-eng]
```

Here you need to choose your device `cm_[your device]-userdebug`, for example if you wish to build for the OnePlus One you would type `cm_bacon-userdebug` or `9`.

#### For Halium-7.1

The `breakfast` command is used in LineageOS 14.1 and above (and therefore halium-7.1) to set up the build environment for a specific device. It is easy to use. Simply ensure that you're in BUILDDIR and run the following:

```
breakfast [codename]
```

Breakfast will attempt to find your device, set up all of the environment variables needed for building, and give you a summary at the end. You don't need to worry about any of this, unless it fails.

### 1.3.2 Modify the kernel configuration

Halium uses systemd as the init system. This requires various specific kernel configurations.

To check which config options needs to be adjusted we use mer-kernel-check utility provided by mer-hybris:

```
git clone https://github.com/mer-hybris/mer-kernel-check
cd mer-kernel-check
./mer_verify_kernel_config <path to kernel configuration>
```

If you don't know the path to your kernel config run `grep "TARGET_KERNEL_CONFIG" device/<VENDOR>/<CODENAME>/BoardConfig.mk`. It should be in `arch/arm/configs/<CONFIG>` or `arch/arm64/configs/<CONFIG>` depending on the architecture of your device.

---

**Todo:** Mention that the config parameters CONFIG_IKCONFIG and CONFIG_IKCONFIG_PROC need to be set to y, otherwise Halium wont boot (or add them to the check script

---

### 1.3.3 Include your device in fixup-mountpoints

Fixup-mountpoints replaces the aliases of block device nodes in `/dev/block/by-name` with their literal nodes under `/dev/block`. This prevents issues caused by `by-name` not being populated by systemd.

First check if the codename of your device is already included in the `<BUILDDIR>/halium/hybris-boot/fixup-mountpoints` script.

If it's not already included, you will need to add it. Your device should be running LineageOS or another ROM where you can get root access over ADB.

1. Find the fstab file for your device. For my Moto G5 Plus, this was `fstab.qcom` in `device/motorola/potter/rootdir/etc`
2. Enable adb root access
3. Create the skeleton for your device in fixup-mountpoints, right before the `*)`:

```
"[codename]")
    sed -i \
        [replacements, one per line]
        "$@"
    ;;
```

4. For every line in fstab where the type is not `auto`, `emmc` or `swap`, run `readlink -f [src]` on the target device over ADB. `[src]` is the leftmost colum in fstab.
5. Write all of our replacements, one for every mountpoint. Here's the bones of one:

```
-e 's [src] [return] ' \
```

Replace [src] with what you input into readlink and [return] with what it returns. The space after [return] is important. The build fails without it.

---

**Note:** Be careful to ensure that your indentation is the same as other devices! The `"[codename]")` line should be indented by four spaces, and everything below it should be indented as shown.

---

### 1.3.4 Building the system.img and hybris-boot.img

Halium will use the mkbootimg tool for creating the boot image. In most cases it is not on the local harddisk, so it can be built by issuing:

```
mka mkbootimg
```

To build the `system.img` and `hybris-boot.img` - required for Halium - use the following commands:

```
mka hybris-boot
mka systemimage
```

---

**Note:** If you use `make` and not `mka` it is recommended to set `-j[num]` to do parallel building, which reduces build time. Replace `[num]` with the number of threads in your system plus 2.

---

If you get any errors, jump down to *Documented errors*. Otherwise, continue on to *Next steps*.

### 1.3.5 Documented errors

If you receive errors while building Halium, check the following documents to see if there is a documented solution.

#### Common kernel build errors

These are the `hybris-boot` build errors most commonly seen in the Halium community. If you have found and fixed a new error, please document the steps that you have taken to fix it.

#### Implicit declaration of 'kvfree'

If you receive something similar to the following error:

```
../../../../../../kernel/[...]/[...]/security/apparmor/apparmorfs.c: In function 'aa_
↪simple_write_to_buffer':
../../../../../../kernel/[...]/[...]/security/apparmor/apparmorfs.c:110:3: error:␣
↪implicit declaration of function 'kvfree' [-Werror=implicit-function-declaration]
kvfree(data);
```

Apply the patch nick kvfree() from apparmor.

### 'kuid_t' (sdcardfs, cgroup) error

Example of the error:

```
../../../../../../kernel/lenovo/msm8916/kernel/cgroup.c:2138:37: error: invalid␣
↪operands to binary != (have 'kuid_t' and 'kuid_t')
if (current != task && cred->euid != tcred->uid &&

../../../../../../kernel/lenovo/msm8916/kernel/cgroup.c:2139:18: error: invalid␣
↪operands to binary != (have 'kuid_t' and 'kuid_t')
    cred->euid != tcred->suid)
```

Set `CONFIG_USER_NS` to `n` in your defconfig.

### Firmware class error

Example of the error:

```
../../../../../../kernel/lenovo/msm8916/drivers/base/firmware_class.c: In function '_
↪request_firmware':
../../../../../../kernel/lenovo/msm8916/drivers/base/firmware_class.c:1226:38:␣
↪warning: passing argument 2 of 'fw_load_from_user_helper' from incompatible pointer␣
↪type
error, forbidden warning: firmware_class.c:1226
```

Set `CONFIG_FW_LOADER_USER_HELPER` to `y` in your defconfig.

### ECRYPTFS error

Example of the error:

```
 ../../../../../../kernel/lenovo/msm8916/fs/ecryptfs/file.c: In function 'ecryptfs_
↪readdir':
 ../../../../../../kernel/lenovo/msm8916/fs/ecryptfs/file.c:130:16: error: assignment␣
↪of read-only member 'actor'
buf.ctx.actor = ecryptfs_filldir;
```

Apply this patch from bullhead.

### Common system build errors

These are the `systemimage` build errors most commonly seen in the Halium community. If you have found and fixed a new error, please document the steps that you have taken to fix it.

### signapk.jar missing

```
error: 'out/host/linux-x86/framework/signapk.jar', needed by 'out/target/product/
↪kenzo/obj/APPS/TimeService_intermediates/package.apk', missing and no rule to make␣
↪it
```

All APKs and Java libraries should be removed from the Makefiles in `vendor/[manufacturer]/[device]` and `device/[manufacturer]/[device]`. The setup script will take care of this for you normally. In case you want to do it manually, simply commenting them out is enough, but you can remove them entirely. An example of these changes can be found at remove APKs on Lyudmila17/android_device_motorola_athene.

Also check the vendor folders of your dependencies.

#### HYBRIS_BOOT_PART and HYBRIS_DATA_PART

```
find: 'device/*/generic': No such file or directory
find: 'device/unknown': No such file or directory
find: 'device/android': No such file or directory
halium/hybris-boot/Android.mk:67: ********************  /boot appears to live on␣
↪ERROR: *fstab* not found
halium/hybris-boot/Android.mk:68: ********************  /data appears to live on␣
↪ERROR: *fstab* not found
halium/hybris-boot/Android.mk:71: *** There should be a one and only one device entry␣
↪for HYBRIS_BOOT_PART and HYBRIS_DATA_PART.
```

Make sure you run the commands *here* before trying to build. The Makefile depends on the environment set up immediately before by envsetup.sh; so if running in a build environment such as Emacs, be sure to set your compile command to something like "source build/envsetup.sh && breakfast [codename] && make [target]"

#### Flex locale error

```
[ 19% 2365/12156] Lex: checkpolicy <= external/selinux/libsepol/cil/src/cil_lexer.l
FAILED: /bin/bash -c "prebuilts/misc/linux-x86/flex/flex-2.5.39 -o/home/peter/docs/
↪devel/halium.amami/out/host/linux-x86/obj/STATIC_LIBRARIES/libsepol_intermediates/
↪cil/src/cil_lexer.c external/selinux/libsepol/cil/src/cil_lexer.l"
flex-2.5.39: loadlocale.c:130: _nl_intern_locale_data: Assertion `cnt < (sizeof (_nl_
↪value_type_LC_TIME) / sizeof (_nl_value_type_LC_TIME[0]))' failed.
Aborted (core dumped)
```

Seems to be a problem with locales and the prebuilt *flex*. You can avoid this by using the "C" locale while building:

```
LANG=C make systemimage
```

If your error is not in this list, please *contact us for help*.

### 1.3.6 Next steps

Now that you have `hybris-boot.img` and `system.img` built, let's install them along with the reference rootfs to test functionality.

## 1.4 Install Halium

At this point, you've set up your build device, downloaded all of the sources for device enablement, and built `hybris-boot.img` and `system.img`. Now, we'll walk you through installing your build along with a distro.

### 1.4.1 Fully halium-integrated distributions

You may choose any of the following distributions to install with the scripts and sources provided by Halium. We recommend you start with the Halium reference rootfs for testing.

#### Halium reference rootfs

Once you have built the system.img from the android tree, you can download and install the rootfs using the `halium-install` script from the halium-scripts repository.

#### Install hybris-boot.img

First, boot your device into its bootloader. This is normally done by holding Power+Volume Down, but it can be different on each device.

Next, simply execute the following command:

```
fastboot flash boot [path/to/]hybris-boot.img
```

If you're in BUILDDIR, `hybris-boot.img` will be located at `out/target/product/[codename]/hybris-boot.img`.

#### Install hybris-boot.img on Samsung devices

Samsung devices cannot be flashed using fastboot. Instead, the device needs to be brought into "Download Mode". This can be achieved in two ways:

1. By manually by pressing Vol-Down, Home and Power button until the green warning text appears. Then press Vol-Up as instructed.
2. By issuing the command:

   ```
   adb reboot download
   ```

Once in download mode the necessary tool depends on platform:

On Windows, proceed with the Odin flashing tool which takes image files wrapped in tgz format.

On Linux, use the Heimdall flashing tool. Heimdall, when used in command line mode, can handle plain .img files.

On recent devices, only newer versions of Heimdall should be used. Ubuntu´s ppa holds an older version of Heimdall. Do not use this. Instead, build it from source:

```
git clone https://github.com/Benjamin-Dobell/Heimdall.git
```

For building instructions, consult the README.

Note: Often you will find instructions on Samsung ROMs saying that you need to obtain a PIT file before flashing a device. This is not required and could in fact soft-brick your device. The PIT file is the partition table and is only required when repartitioning. This is normally not necessary and using downloaded PIT files is risky.

Heimdall will always reboot by default after flashing. Using the –no-reboot option will leave the connection in a strange state. Therefore, after flashing a .img file it is not possible to immediately push a second one. Also, Heimdall is incapable of rebooting directly into recovery mode.

The command for flashing is:

```
heimdall flash --BOOT hybris-boot.img
```

### Install rootfs and system.img

Currently the latest rootfs is available at bshah's personal server: Link

You can use the halium-install script as below, when the device is connected in recovery mode:

```
halium-install <path to rootfs tarball> <path to android system.img>
```

This will do the following:

- Convert the rootfs tarball into ext2 rootfs.img
- Convert system.img into the ext4 mountable image from android sparse image.
- Push the both images to /data partition

### Install rootfs and system.img (alternative method)

If the method above failed for some reason, you could try the `halium-install` script from the JBB's repository.

You can use the halium-install script as below, when the device is connected in recovery mode:

```
halium-install -p halium <path to rootfs tarball> <path to android system.img>
```

If you have trouble getting this to work, use the `-v` option to get verbose output.

### Debugging

Now that you have this installed, move on to *Debug Halium* to test your port's functionality.

### Plasma mobile

In order to install Plasma mobile you perform basically the same steps as for the *Halium reference rootfs*. This section only highlights the relevant differences.

### Install rootfs and system.img

Use the plasma mobile rootfs. Download the latest version from here: https://images.plasma-mobile.org/rootfs/.

If you have a "CAF" device (Code Aurora Forum), then you must use the rootfs from here: https://images.plasma-mobile.org/caf-rootfs/. Generally speaking, if it is a Qualcomm device and it is not a Nexus, then it is CAF.

### Debugging

Instead of connecting with *root* you use *phablet*:

```
ssh phablet@10.15.19.82
```

**Todo:** Make sure phablet password is set either by using chroot from recovery or halium-install from JBBgameich.

For general debugging tips refer back to reference rootfs instructions. Some plasma mobile specific tips are below.

**Todo:** Document PM specific debug instructions. Some lose notes copied from irc log:

- run test_XYZ with sudo
- pgrep kwin_wayland
- journalctl | grep simplelogin

### 1.4.2 Halium-modified distributions

These distributions require changes to the source or use different scripts than those provided by Halium. We recommend you only try these after your build tests correctly with the reference rootfs.

**Todo:** Distributions like Ubuntu Touch and LuneOS go here. Links to their respective documentation should be sufficient, but please place them inside the toctree above.

## 1.5 Debug Halium

The following subsections should help you through the porting process.

Keep in mind, every port is a bit different. You might be following this from top to bottom and even skip steps. Or, you might end up going back and forth between the sections while you improve your port.

### 1.5.1 Early Init

Hybris-boot offers a telnet service that you can use to debug the very early init of your port. This service will be exposed if the Halium system has failed to boot for any reason.

#### Determining if this is needed

While bringing the USB network interface up, the boot image will write a few debug messages. These debug messages are communicated via a clever hack of resetting the serial number of the usb connection.

The steps in detail are:

- Execute this command to watch the changes in the usb serial number:

```
while : ; do lsusb -v 2>/dev/null | grep -Ee 'iSerial +[0-9]+ +[^ ]' ; done | uniq
```

- Boot your newly built image
- Watch the output of the lsusb command above. It will put out lines like this:

```
iSerial                 3 01234567
iSerial                 3 Mer Debug setting up (DONE_SWITCH=no)
iSerial                 3 Mer Debug telnet on port 23 on usb0 192.168.2.15 - also␣
↪running udhcpd
```

If you get a line similar to the last two above, Telnet is running. Continue to the next section to debug it.

If you instead get the line `GNU/Linux devices on rndis0 10.15.19.82`, the system has booted successfully. Move to *Logging in* to continue your debugging.

### Debugging via telnet

- **Determine the name of the usb network device on your desktop:** `dmesg | tail`. You're looking for a line similar to this:

  ```
  [ 1234.123456] rndis_host 1-7:1.0 enp0s20f0u7: renamed from usb0
  ```

- In this example shown above, `enp0s20f0u7` is the usb network device name. Use this for the USBNETWORK below
- Check if the usb network device has a MAC address assigned:

  ```
  $ ip address show dev USBNETWORK
  6: USBNETWORK: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default␣
  ↪qlen 1000
   link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
  ```

  If it shows the link/ether address `00:00:00:00:00:00` as shown above, you will have to manually assign the MAC address:

  ```
  ip link set USBNETWORK address 02:01:02:03:04:08
  ```

  You can set any MAC address you want, it just needs to be a valid MAC address.

- Configure usb networking:

  ```
  sudo ip address add 192.168.2.1 dev USBNETWORK
  ip address show dev USBNETWORK
  sudo ip route add 192.168.2.15 dev USBNETWORK
  ping -c 2 192.168.2.15
  ```

- Connect with telnet: `telnet 192.168.2.15`

Now you have terminal access to the system running from initramfs. The first command to run once you're logged in is `cat diagnosis.log` to see if it has any hints.

### Forcing debug mode

If the device simply reboots when trying to boot and does not bring up telnet, you may build and use the `hybris-recovery.img` file to attempt to force a shell to come up. To do this, set up your build tree and issue the following command:

```
mka hybris-recovery
```

The file will be in the standard $OUTDIR. Simply flash it in the same way you did for `hybris-boot.img`.

#### Common errors

##### The device reboots after leaving hybris-recovery

If your device reboots after you leave hybris-recovery by running `echo continue > /init-ctl/stdin`,there could be something wrong with your lxc container. Try the following in the hybris-recovery shell to disable it for the moment:

```
TERM=linux HOME=/root PATH=/sbin:/bin:/usr/sbin:/usr/bin:$PATH chroot target/ /bin/
↪bash
systemctl disable lxc@android
systemctl mask lxc@android
exit
echo continue > /init-ctl/stdin
```

You should then be offered SSH. See *Logging in* for more details. Once you are logged in, you can try to troubleshoot what is causing your reboots by running `systemctl unmask lxc@android && systemctl start lxc@android`.

##### Bootloop: "Too many levels of symbolic links" when leaving hybris-boot

```
systemd[1]: Failed to determine whether /sys/fs/cgroup is a mount point: Too many␣
↪levels of symbolic links
systemd[1]: Failed to mount cgroup at /sys/fs/cgroup/systemd: NO such filE oR␣
↪directory
systemd[1]: Freezing execution.
```

There's a patch on hammerhead kernel from Linus Torvalds you need to apply: Fix /proc/<tid>/fdinfo/<fd> file handling.

##### The device reboots with hybris-recovery

There are several cases in which telnet will not be exposed, such as when the device fails to load the kernel or initramfs, or when an in-kernel driver decides to cause a kernel panic very early. In this case, the phone will almost immediately reboot when it starts, even when using `hybris-recovery`. You may be able to *read the previous boot's kernel message buffer*. Please have that ready and *contact us for help*.

##### None of these describe my issue

You may be able to *read the previous boot's kernel message buffer*. Please have that ready and *contact us for help*.

### 1.5.2 Logging in

Once booted, the reference rootfs will offer an SSH connection that you can use to run tests or otherwise debug the system. Before you're able to sign in, though, you will need to change the root user's password.

To change the root user's password, reboot into TWRP and get access via `adb shell`. Then, perform the following commands:

```
mkdir /a
mount /data/rootfs.img /a
chroot /a /bin/bash
. /etc/environment
passwd
```

You will be prompted to change the password. Once finished, clean up:

```
exit
umount /a
rmdir /a
sync
```

You may now reboot your device into Halium.

Once your device is booted, you should see that you have a new network interface again. Look at the output of `ip link show`, the name will be e.g. enp0s29u1u1. rndis0 is not working on most kernels. Assign it a fake MAC address (if the default one is all 00s) and an IP of 10.15.19.100:

```
   ip link set <devicename> address 02:11:22:33:44:55
   ip address add 10.15.19.100 dev <devicename>
   ip link set <devicename> up

Once finished, you should be able to do the following to log in::

   ssh root@10.15.19.82
```

### 1.5.3 Add udev rules

You must create certain udev rules to allow some tests access to the hardware.

There are 2 different ways to do this:

#### On the device:

Log in on the device, run the following command, replacing [codename] with your device's codename:

```
cat /var/lib/lxc/android/rootfs/ueventd*.rc | grep ^/dev | sed -e 's/^\/dev\///' |␣
↪awk '{printf "ACTION==\"add\", KERNEL==\"%s\", OWNER=\"%s\", GROUP=\"%s\", MODE=\"
↪%s\"\n",$1,$3,$4,$2}' | sed -e 's/\r//' >/etc/udev/rules.d/70-[codename].rules
```

Now, reboot the device.

#### On the host:

Alternatively, if you want to create the file with the udev rules on your "host" machine you can run the following command inside your halium folder, replacing [codename] with your device's codename:

```
cat out/target/product/[codename]/root/ueventd*.rc | grep ^/dev | sed -e 's/^\/dev\///
↪' | awk '{printf "ACTION==\"add\", KERNEL==\"%s\", OWNER=\"%s\", GROUP=\"%s\",␣
↪MODE=\"%s\"\n",$1,$3,$4,$2}' | sed -e 's/\r//' > 70-[codename].rules
```

You can then use either `ssh` or `adb push` to push the file to the right location:

SSH:

```
cat 70-[codename].rules | ssh 10.15.19.82 'cat > /tmp/udev.rules'
ssh 10.15.19.82
sudo cp /tmp/udev.rules /etc/udev/rules.d/70-[codename].rules
```

ADB: Reboot into recovery. If your rootfs is mounted at `/data/halium-rootfs/` you can push it with:

```
adb push 70-[codename].rules /data/halium-rootfs/etc/udev/rules.d/
```

#### Send upstream:

In order for other users to be able to use the udev rules send a pull request to the lxc-android repository. Similar to: https://github.com/Halium/lxc-android/pull/12/files

### 1.5.4 Wifi

#### Enabling Wi-Fi hardware

#### Qualcomm

Wifi is fairly easy to get going on most Qualcomm devices. The following should enable your device's Wi-Fi hardware:

```
echo 1 > /dev/wcnss_wlan
echo sta > /sys/module/wlan/parameters/fwpath
```

#### Errors

#### If the first command line fails with `/dev/wcnss_wlan: Bad address`

This is not an error condition; don't get put off — try the second command.

#### If `/sys/module/wlan` is missing (not found)

Try to load the `wlan` kernel module first:

```
insmod /system/lib/modules/wlan.ko
```

If `insmod` fails with `Required key is not found`, you could temporarily disable all the `CONFIG_MODULE_SIG**` options in your `defconfig` and then re-build `hybris-boot` (or `halium-boot`) and `system.img`. **Beware: this disables kernel module signatures (a security feature) — consider fixing it later.**

If `insmod` fails with `Invalid module format`, then you need to ensure that you build your `hybris-boot` (or `halium-boot`) and `system.img` with the same kernel configuration (e.g. perhaps you forgot to re-install `system.img` after disabling kernel module signatures above, it contains kernel modules).

#### Broadcom bcmdhd

It is recommended to build Broadcom drivers as a module since that will ensure use of the device's MAC address.

In the kernel defconfigs, make sure these settings are set:

```
CONFIG_MODULES=y
CONFIG_BCMDHD=m
```

Then add this to your device's init.rc file, it's recommended to set this in early stages to avoid race condition with network manager (`on post-fs-data` is a good place for this):

```
insmod /system/lib/modules/bcmdhd.ko
```

#### Testing Wi-Fi functionality

You may run the following command to see if your Wi-Fi hardware has come up:

```
nmcli d
```

Your device should show up as `wlan0` and have a state of "Disconnected" when it is ready. At that point, run the following command to enter an interface that you can use to connect to Wi-Fi:

```
nmtui
```

Once you are connected to Wi-Fi, try pinging an Internet device:

```
ping 8.8.8.8
```

If all of this is successful, you have successfully brought up your Wi-Fi hardware. If not, check your device's *Logcat* for possible errors.

#### Common errors

##### Kernel 3.10 ping: `socket: Permission denied`

This error is most common on devices which shipped with Linux kernel 3.10. To resolve it, apply the following patches:

1. Introduce the SECURITY_ANDROID_GID_CAPABILITIES option
2. Select SECURITY_ANDROID_GID_CAPABILITIES when enabling paranoid network
3. Enable the CONFIG_SECURITY_ANDROID_GID_CAPABILITIES option

### 1.5.5 Lights

Lights is one of the simple parts to get going in Halium.

#### Tests

```
test_lights
```

The device's light should start flashing now.

### 1.5.6 Graphics

Graphics is an essential part of Halium. Halium uses libhybris to make Android's bionic based hardware adaptations layer usable with glibc systems.

#### Tests

Here is some tests to test the graphics stack

```
EGL_PLATFORM=hwcomposer test_hwcomposer
```

```
test_egl
```

```
test_egl_config
```

```
test_glesv2
```

**Todo:** Add tests using more heavy graphics applications using Wayland (mir)

#### Common errors

#### EGL_BAD_SURFACE

If these tests fail but some others succeed (lights, vibrator, Wi-Fi), you may need software which is built for Code Aurora Forum's Android for MSM (generally nicknamed "CAF" for short) rather than AOSP.

If this is the case, you'll encounter errors similar to the following in *Logcat*:

```
W Adreno-GSL: <gsl_ldd_control:475>: ioctl fd 8 code 0xc0140933 (IOCTL_KGSL_TIMESTAMP_
↪EVENT) failed: errno 22 Invalid argument
W Adreno-GSL: <ioctl_kgsl_syncobj_create:2984>: (9, 1, 62845) fail 22 Invalid argument
W Adreno-EGLSUB: <SwapBuffers:1339>: gsl_device_3d_add_fence_event failed
W Adreno-EGL: <qeglDrvAPI_eglSwapBuffers:3890>: EGL_BAD_SURFACE
```

On the reference rootfs, you can get this software from the Halium repository by running the following commands:

```
. /etc/environment
echo "deb http://repo.halium.org/caf xenial main" >> /etc/apt/sources.list.d/halium-
↪caf.list
apt-get update
apt-get dist-upgrade
```

Non-reference distributions will need to provide software which is compatible with CAF Android trees.

#### Reading pm4 microcode failed

This error may appear in *Logcat* when running these tests:

```
kgsl kgsl-3d0: |_load_firmware| request_firmware(a330_pm4.fw) failed: -2 ... kgsl␣
↪kgsl-3d0: |adreno_init| Reading pm4 microcode failed a330_pm4.fw
```

To resolve this, toggle `CONFIG_FW_***LOADER` in your kernel config.

#### test_hwcomposer failure on aarch64 device with armhf rootfs

On an armhf only rootfs, test_hwcomposer will fail despite doing the above on 64-bit devices running kernel 3.18.

To work around the issue, apply this patch to your device's kernel repository.

### 1.5.7 Vibrator

Vibrator is one of the simple parts to get going in Halium, and is often used as to goto test to check if libhybris linker works as expected.

#### Tests

```
test_vibrator
```

The device should now vibrate

### 1.5.8 Debugging the Android userspace

Debugging the Android userspace

#### lxc-android

lxc-android is the container in which the Android userspace is running. You can check that it is started with the following command:

```
systemctl status lxc@android
```

LXC needs some kernel config to make sure it runs correctly. Check that you have all the needed options by running the following command on the device:

```
lxc-checkconfig
```

All option except `User namespace` need to be the green word *enabled*. If one of the options is a yellow *missing* or a red *required*, then you need to change the kernel config, rebuild hybris-boot and check the status again.

---

**Note:** I was getting the following error, which I didn't understand: `lxc-start: utils.c: mkdir_p: 254 Invalid argument - failed to create directory '/sys/fs/cgroup/net_cls//lxc/android'` It appears like LXC is expecting some functionality that doesn't work yet in Linux 3.4. Counter-intuitively I had to disable the following functionalities in the defconfig to make it work:

```
CONFIG_NET_CLS_CGROUP=n
CONFIG_NETPRIO_CGROUP=n
```

---

#### Logcat

Logcat is a tool that reads the Android user space logs. This includes all services that should be running in Halium. You can run it at any time with the following command:

```
/system/bin/logcat
```

For radio (Wi-Fi, GSM, LTE) logs, you can add a flag:

```
/system/bin/logcat -b radio
```

If you're not able to run this command for any reason (for example, because you're running an armhf rootfs on an arm64 device), you can try to run it inside the Android container via `lxc-attach`:

```
lxc-attach -n android -- /system/bin/logcat
```

You may similarly use this to run any binary inside the Android system. Simply replace the command after the two dashes.

### dmesg

Even though Android logs do not normally end up in dmesg, early initialization of Android and kernel output ends up here:

```
dmesg
```

### strace

For cases where the log files do not reveal sufficient detail, a `strace` can be helpful. This is how you get a strace for the example of `test_hwcomposer`:

```
EGL_PLATFORM=hwcomposer strace test_hwcomposer
```

### backtrace

Another debugging technique is to investigate the backtrace when a program crashes. This is how you get a backtrace for the example of `test_hwcomposer`:

```
EGL_PLATFORM=hwcomposer gdb test_hwcomposer
```

This will start the interactive debugger `gdb`. At the prompt of `gdb` you enter `run`. Now the program is executed and you wait for it to crash. Then you enter `bt full`. This will give you the full backtrace of what the program was trying to execute at the moment of the crash.

In order to make the backtrace most useful you want to ensure that you have debug symbols installed for the program you are debugging.

Firstly, let's fix the `PATH` variable which is currently missing `/sbin` on the reference rootfs:

```
export PATH=$PATH:/sbin
```

Secondly, do install debug symbols for libc:

```
apt install libc6-dbg
```

Thirdly, install whichever package contains the debug symbols for the program in question. Typically it is in a package with a name similar to the one containing the program and ending in `-dbg`. For the example of `test_hwcomposer` you want:

```
apt install libhybris-dbgsym
```

If `gdb` reports "(no debugging symbols found)", then you are still missing debug symbols, look further for the relevant package.

**Todo:** Document the debugging of libhybris: https://wiki.ubuntu.com/Touch/Core/UbuntuDebugAndroid

**Todo:** Document how to deal with firmware partitions.

For example xLeEco Le Max2, codename "x2" has a firmware partition where the vendor blobs are stored. Initially lxc@android would not start. The resolution was roughly:

- no need for a vendor blobs repository in the manifest
- determine firmware partition name
- ensure fix-mountpoints takes it into account
- reflash android to ensure the blobs are in the partition
- reflash halium

See http://logs.nslu2-linux.org/livelogs/halium/halium.20180430.txt

### 1.5.9 USB Network tethering

It is possible to connect the device to the internet via usb network tethering. This can be helpful when you haven't gotten Wi-Fi working (yet).

The instructions below assume you're running Ubuntu >=16.04. Earlier versions of Ubuntu or other distributions will probably work similarly if `iptables` is installed.

You need three things:

- The name of your computers network device which connects to the internet. You can check `ip link` to determine this. Likely it is something like `wlan0` or `eth0`. We'll call this `[INTERNET]`.
- The name of your computers usb network device which connects to the "rndis" network. This is the same as described for the *telnet debugging access*. We'll call this `[USBNETWORK]`.
- Lastly, you need your computers IP address on this network. If you are connecting with *telnet* then this is `192.168.2.1`. If you are connecting with *ssh* then this is `10.15.19.100`.

With this information you run the following commands:

```
sudo sysctl net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o [INTERNET] -j MASQUERADE
sudo iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i [USBNETWORK] -o [INTERNET] -j ACCEPT
```

Now, run the following command as root on your device:

If you're in initrd debug (telnet):

```
route add default gw 192.168.2.1
```

If you're in the real rootfs (ssh):

```
ip route add default via 10.15.19.100
```

Then try `ping 8.8.8.8` from the phone. The pings should go through.

To add an DNS server on your device, edit `/etc/resolv.conf` and add the line `nameserver 8.8.8.8`. This is NOT the recommended way to add a nameserver on Debian, but it works reliably when you can't depend on other options. Your changes will be overwritten on reboot, so you will need to perform them again.

You should now be able to `ping debian.org` and generally have internet access on the device.

### 1.5.10 Reading kernel logs

To find out what happened during an unsuccessful boot you can check the kernel log files. These can usually be retrieved after rebooting the device into another, working system. Generally, your recovery image will be the "working system".

#### Linux kernel <=3.4

Until Kernel 3.4, Android developers created a driver that the Linux kernel may use to store its kernel message buffer in a special persistent location in RAM. This information is preserved over a reboot but not a hard power cycle (such as holding the power button or powering off). This location is preserved to the system in `/proc/last_kmsg`

To use `last_kmsg` to debug problems in your port, do the following:

1. Boot your newly built image
2. Wait for it to fail
3. Reboot the device into a working system.
4. Retrieve the kernel log with `adb shell cat /proc/last_kmsg > ~/last_kmsg`
5. Read `~/last_kmsg` and find out what went wrong

If you aren't able to find these files, ensure that the following configs are set in your kernel config:

```
CONFIG_ANDROID_RAM_CONSOLE=y
CONFIG_ANDROID_RAM_CONSOLE_ENABLE_VERBOSE=y
```

#### Linux Kernel >3.4

After Kernel 3.4, most Android vendors have used the upstream `pstore` and `ramoops` drivers to store kernel logs after a panic.

You can get these logs by following these steps:

1. Boot your newly built image
2. Wait for it to fail
3. Reboot the device into a working system.
4. Find and retrieve the logs from `/sys/fs/pstore/console-ramoops`. The file may be named slightly differently, but will be in this directory.

If you aren't able to find these files, ensure that the following configs are set in your kernel config:

```
CONFIG_PSTORE=y
CONFIG_PSTORE_CONSOLE=y
CONFIG_PSTORE_PMSG=y
CONFIG_PSTORE_RAM=y
```

Most devices require that the kernel panics in order to offer these logs. You can cause a kernel panic by running these commands as root:

> **Warning:** Yes, these commands really do cause a kernel panic. Don't run them on your production machine.

```
echo 1 > /proc/sys/kernel/sysrq
echo c > /proc/sysrq-trigger
```

### 1.5.11 References

The hybris-boot image is based on work from the Sailfish OS and the Sailfish Hardware Adaptation Development Kit porting guide contains valuable tips.

## 1.6 Distribution

### 1.6.1 Reference rootfs

To make development and porting easier the halium team provides the reference rootfs. It is based on Ubuntu 16.04 and includes the following:

- systemd
- lxc
- lxc-android
- libhybris
- ofono
- ofono-scripts

This reference rootfs is built using the live-build tool. Documentation on how you can build it locally using rootfs-builder is available at https://github.com/halium/rootfs-builder/tree/ubuntu

---

### 1.6.2 Deployment

There are various parts which need to be deployed on the device:

- boot.img
- rootfs
- udev rules
- lxc container configuration

- system.img
- vendor.img (for newer devices)
- android ramdisk

### boot.img

The boot.img is based on hybris-boot. This boot.img is built inside the android tree.

boot.img has really simple functionality: Mount the data partition, and the rootfs.img inside to /target,

```
mount $DATA_PARTITION /data
mount /data/rootfs.img /target
```

and switch to the target rootfs and run the init (systemd)

```
exec switch_root /target $INIT --log-target=kmsg &> /target/init-stderrout
```

### rootfs.img

The rootfs is provided by the distribution in tar.gz format. The installation tool extracts the contents and puts them into the ext2 or ext4 rootfs.img. This rootfs.img will be mounted by the hybris-boot later on.

Following are the minimal requirements of the such a rootfs

- systemd as the main init
- lxc

Depending upon required functionality, you may need

- ofono (for calling and mobile data)
- libhybris
- NetworkManager
- Pulseaudio

### udev rules

The rootfs.img contains the udev rules. They are device specific and generated from the ueventd*.rc file from the device tree. Instructions for getting these rules installed can be found at *Add udev rules*.

You can either,

1. copy them to /lib/udev/rules.d/ during the initial installation of the rootfs or
2. deploy all the supported udev rules in /usr/lib/lxc-android/, and have a systemd service to copy the device udev rule before starting udev service.

### lxc container configuration

This is most important bit. It starts the android init. This is required to start the android binary daemons etc.

There are two parts of the lxc container configuration:

- config

- pre-start.sh

They are provided at : https://github.com/Halium/lxc-android/blob/master/var/lib/lxc/android/

The configuration file specifies the following options:

- lxc.rootfs
- lxc.network.type
- lxc.devttydir
- lxc.tty
- lxc.pts
- lxc.arch
- lxc.cap.drop
- lxc.pivotdir (deprecated)
- lxc.hook.pre-start
- lxc.init_cmd
- lxc.aa_profile (optional, only for distribution using apparmor)
- lxc.autodev

The pre-start hook is used to extract and configure the android rootfs before booting into it.

#### system.img and vendor.img

These are the android libraries and the vendor binary blobs. They are generated by building the android tree and they are deployed to the userdata partition at installation time.

#### Android rootfs

The android rootfs is provided by halium. It is generated by the android build system. The android rootfs contains what is usually contained inside the initrd of android's boot.img. This is extracted by the android lxc container's pre-start hook before starting the container. It is located at /system/boot/android-ramdisk.img.

---

### 1.6.3 Startup sequence

- fastboot starts the kernel and loads the initrd
- initrd mounts the userdata partition and rootfs.img from it
- After mounting rootfs.img it will start the systemd init from the rootfs
- The rootfs is expected to mount the /system, /vendor and other android mount points before local-fs.target
- After the local-fs target, the lxc container is started
- lxc pre-start hook will bind mount the mounted android partitions inside the android rootfs
- Once the android container is started the host system will start udev and other system daemons
- At this point the scope of halium is over and userspace services like sddm, mir, lipstic etc can be started

## 1.7 About this chapter

In this section, we want to gather information about

- The exact hardware built into our phones and tablets
- The status of ports, that is:
  - Which OS is available in what version and to which degree of functionality
  - Which kernel is available? What drivers are used for what part?
  - Where are the sources for all this? What is their license?
- Where can we reuse drivers/kernel patches/backports?

### 1.7.1 Where should I put my information?

The documentation has a logical structure so that every specific group of devs we are trying to help here can access the info that is meant for them without having to read the other parts.

### 1.7.2 Why is this important?

Halium is trying to reduce the workload for different OS teams by developing a common base. The deepest root in all of the OS systems, even the official ones, is the kernel, which is based upon the linux mainline kernel. For Halium to work, certain features need to be implemented into this kernel, for example the ability to use lxc.

Therefore it is of great interest to us to have a kernel for each device that has all the needed parts. The sense behind mainlining is that **you only care about the specific hardware you are working on**. All other features that are implemented by other people will just be handled mainline and if your code is mainline too, upon a kernel update, you can just use the new one with new features from other people, without having to reimplement your driver into this new kernel. So if e.g. lxc is updated or apparmor and we are working on an older kernel, at some point the rootfs we are working with will require this new version and we need to do all the implementation of our work to a new kernel.

By documenting things **that will never change** like the hardware built into a phone, the kernel version a driver was mainlined, or general mainline status of hardware parts, we reduce the risk to document something that will change before it is ever looked at.

On the other hand, comparing the hardware of devices shows, that many parts are used across manufacturers, so many drivers may, at least partially, be cloned.

With documenting these things at a central point, we may make connections easier visible and by this enable kernel devs or newcomers to write/modify drivers and mainline them.

For our porters, we can gather info about available ROM versions, where the sources and proprietary blobs (if available) are located, and perhaps what challenges/problems may occur if any are known already.

### 1.7.3 Who is this for?

This sub section of the documentation is meant for

- **porters** as a first quick reference about the status for certain hardware parts (drivers) and whole devices (custom kernels & OSes)
- **community members** who want to help by documenting
- **kernel devs** as a reference which hardware is in what device, if there are open source drivers or where the official drivers are located

## 1.8 Device Overview

These pages detail devices that

- ( A ) **have a running Halium Image**
- ( B ) **had a Halium image in the past**
- ( C ) **users want to see ported**

All of these should only be document in the exact same way here and **only an overview shall be given on this page** everything that goes into more detail should please be posted on a device specific sub page!

### 1.8.1 Codename - Template Device

**//Disclaimer:** Please remove everything that is just part of this template and not actually about the device before submitting this into the official wiki, thank you! **//**

Here only one or two sentences. What makes the device special? Is it a reference device?

#### Status

#### Halium

Of course we are mainly interested in Halium status. Therefore please state

- Is someone already working on Halium for this device? If YES: link to repositories of **device specific files** (usually github repos called "android_device_manufacturer_codename") and **device kernel** (usually a repo beginning with "android*kernel*") and name authors if possible (perhaps link to their github profile)
- Is Halium working for this device? If YES: Name the android base version (5.1 or 7.1) and everything you know about the actual status (what's working, which rootfs, . . . )

#### Distributions

The following entries are just a placeholder, exactly as this sentence.

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS placeholder device page placeholder | android_device_placeholder | android_kernel_placeholder based on vX.Y.Z | ? | ? |
| Ubports placeholder device page placeholder | android_device_placeholder | android_kernel_placeholder based on vX.Y.Z | ? | ? |

#### Kernel & Hardware

#### Mainline (vX.Y.Z as of writing)

Write whether something that is needed for the device is mainline already (switch the version in the heading for what's recent when you write this). This means **device tree source files (.dts) as well as single drivers** (for example only the wifi driver).

#### Cyanogemod based kernels (LOS & UBP)

If other kernels exist, just make up headlines with a name that describe the origin (Cyanogemod, LineageOS, UBports, Canonical, ...) and which is the underlying mainline version. Afterwards describe what's been improved or altered and if possible why or what is still missing.

### Device Specifics

### Guides

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database.

### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

### Useful Resources

If anything might be usefull but didn't fit above you can just throw in some links here.

## 1.8.2 dream2lte - Samsung Galaxy S8+

The Samsung Galaxy S8+ (shortened to S8+) are Android smartphones produced by Samsung Electronics as part of the Samsung Galaxy S series. The S8+ was unveiled on 29 March 2017 and directly succeeds the Samsung Galaxy S7 Edge.

### Status

### Halium

Halium 7.1 is **WIP**. Ported by Ivan Semkin.

### Distributions

| Distribution | Device Specific Files | Device Common Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|---|
| dream2lte on XDA | univer-sal8895/android_device_samsu... | universal8895/android_device_samsung_dream2lte | univer-sal8895/android_kernel_samsung_universal8895 based on v4.4.79 | see de-vice page | see device page |
| dream2lte on Halium project-man-agement | univer-sal8895/android_device_samsu... | vanyasem/android_device_samsung_dream2lte | vanyasem/android_kernel_samsung_universal8895 based on v4.4.79 | see de-vice page | see device page |

#### Kernel & Hardware

#### Cyanogemod based kernels (LOS & UBP)

Both **Halium and LOS share a common base** which is Cyanogenmod. However, the Halium kernel was patched with Halium-specific edits.

#### Device Specifics

#### Guides

Unlocking the bootloader will permanently set your device's KNOX to 0x1, which will disable Knox-related functionality even if you re-lock your device later.

The device doesn't have fastboot, you need to flash it using Heimdall.

To unlock the bootloader enable `OEM Unlock` setting in Developer Options of your ROM.

In order to launch download mode, with the device powered off, hold `Volume Down` + `Bixby` + `Power`.

In order to launch recovery, with the device powered off, hold `Volume Up` + `Bixby` + `Power`.

#### Useful Resources

- Repo Manifest
- TWRP for Samsung Galaxy S8+

### 1.8.3 Bullhead - Nexus 5X

This is supposed to become one of our **reference devices**.

#### Status

#### Halium

Halium is being worked on for this device. You can find the sources in the table below.

**Todo:** Add the manifest for Bullhead

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Bullhead on LineageOS Wiki | LineageOS/android_device_lge_bullhead | LineageOS/android_kernel_lge_bullhead based on v3.10.73 | ? | ? |

#### Kernel & Hardware

#### Mainline Kernel (4.13rc4 as of writing)

There is **no device tree source** (dts) file in the mainline kernel, neither for the device itself nor for the underlying SoC (MSM8992). The general support for Nexus 5X and Nexus 6 landed with 4.10, according to a Phoronix article about Kernel 4.10 ARM support. There is also a short slideshow by Jeremy McNicoll about the initial and ongoing mainlining process.

#### LOS 14.1 Android Kernel (3.10.73)

Unclear whether security bug fixes are implemented into this kernel (long term 3.10 kernel @ kernel.org has version 3.10.107).

#### Device Specifics

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database. For now check the LOS device page linked in the table above.

### 1.8.4 Cedric - Motorola Moto G5

The Moto G5 is a mid-range phone, with a few variants.

The XT1676 variant is special in that it has 3 GB of RAM.

---

**Todo:** Document some other device variants.

---

#### Status

#### Maintainership

| Who | What | GitHub | XDA-Developer | Other Links |
|---|---|---|---|---|
| Olivier | Initial documentation | reivilibre | LambdaPerl | n/a |

#### Halium

No progress to speak of, yet.

### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Unofficial LineageOS 14.1 (GitHub User: Wzedlare) | android_device_motorola_cedric/cm-14.1 | android_kernel_motorola_msm8937/cm-14.1 based on v3.18.49 | The vast majority of things | ? |
| Unofficial LineageOS 15.1 (GitHub User: Wzedlare) | android_device_motorola_cedric/lineage-15.1 | android_kernel_motorola_msm8937/lineage-15.1 based on v3.18.100 | ? | ? |

### Kernel & Hardware

#### Mainline (v4.16-rc7 as of writing, 2018-03-31): Not Mainline

There is no device tree source (dts) file in the mainline kernel, neither for the device itself nor for the underlying SoC (MSM8937). Consult the table below for status on other hardware.

| Kind | Component | Progress |
|---|---|---|
| Chipset/SoC | MSM8937 (msm8937.dts) | Not mainline yet. No known work yet. |
| GPU | Adreno 505 | YES, mainline |
| Image Sensor (Front) | OV5695 (5 MP) | **TODO** |
| Image Sensor (Back) | IMX258 (13 MP) | **TODO** |

Other hardware still needs research.

#### Unofficial LineageOS 15.1 Android Kernel (3.18.100)

Note: the 3.18.x branch of the Linux kernel has been marked as End-Of-Life (EOL).

Available at:

- Device Tree: https://github.com/Wzedlare/android_device_motorola_cedric/tree/lineage-15.1
- Kernel: https://github.com/Wzedlare/android_kernel_motorola_msm8937/tree/lineage-15.1
- Vendor Blobs: https://github.com/Wzedlare/android_vendor_motorola_cedric/tree/lineage-15.1

#### Unofficial LineageOS 14.1 Android Kernel (3.18.49)

I have included this, despite being less updated than the LOS15.1 version, as it (LOS 14.1) is what I use on my phone today.

Available at:

- Device Tree: https://github.com/Wzedlare/android_device_motorola_cedric/tree/cm-14.1
- Kernel: https://github.com/Wzedlare/android_kernel_motorola_msm8937/tree/cm-14.1
- Vendor Blobs: https://github.com/Wzedlare/android_vendor_motorola_cedric/tree/cm-14.1

### Motorola Kernel Sources (?.??.???)

As used on the XT1676 devices. Not sure about the version, would have to check.

https://github.com/MotorolaMobilityLLC/kernel-msm/releases/tag/MMI-NPP25.137-15

Other devices have a different set of sources from Motorola. Research may be required if someone is interested in locating them.

### Device Specifics

### Guides

### Unlocking the bootloader

To unlock the bootloader, you have to provide a code to Motorola's online service who will warn you that your warranty will be void before giving you an unlock code.

Be sure to back up your stock image if you want to return to it, as Motorola do not provide images for this device, despite there being a friendly-looking 'stock ROMs' section on their developer site.

Once your device is unlocked, your boot photograph will be replaced by a warning with hard-coded text over the top of it. You can change the boot photograph but the white text will always remain, so you need to use a boot photograph with white in the correct region to prevent this from being visible.

**Warning:** There have been reports of hard-bricking devices by restoring stock, relocking the bootloader and updating through Motorola channels. Consult xda-developer threads about this issue if this concerns you. (If you don't intend on returning to stock, this is likely not a concern.)

**Todo:** TODO provide a nice link to help users unlock their bootloader

### Access the bootloader

When your device is turned off: Hold down the Volume-Down button whilst you hold down the Power button. Within a few seconds, you should be greeted by the bootloader menu.

Use Volume-Up and Volume-Down to choose between the choices (such as 'Start', 'Recovery', 'Restart Bootloader', amongst others) and press Power to activate your choice.

### Developer Info

The XT1676 had segmentation faults with a particular (kernel?) configuration under LineageOS 14.1 and other custom Android versions. Investigation may be required to track down the cause and solution in the event that the Halium porting effort runs into it again.

### Useful Resources

**Todo:** Provide some useful resources here.

### 1.8.5 Chaozu - BQ Aquaris U

Here only one or two sentences. What makes the device special? Is it a reference device?

#### Status

#### Halium

A port for **Halium 7.1 is being worked on** by LNJ2 and JBBgameich. See the Halium project management issue for chaozu.

#### Distributions

The following entries are just a placeholder, exactly as this sentence.

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS placeholder device page placeholder | android_device_placeholder | android_kernel_placeholder based on vX.Y.Z | ? | ? |
| Ubports placeholder device page placeholder | android_device_placeholder | android_kernel_placeholder based on vX.Y.Z | ? | ? |

#### Kernel & Hardware

#### Mainline (vX.Y.Z as of writing)

Write whether something that is needed for the device is mainline already (switch the version in the heading for what's recent when you write this). This means **device tree source files (.dts) as well as single drivers** (for example only the wifi driver).

#### Cyanogemod based kernels (LOS & UBP)

If other kernels exist, just make up headlines with a name that describe the origin (Cyanogemod, LineageOS, UBports, Canonical, ...) and which is the underlying mainline version. Afterwards describe what's been improved or altered and if possible why or what is still missing.

#### Device Specifics

#### Guides

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database.

#### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

#### Usefull Ressources

If anything might be usefull but didn't fit above you can just throw in some links here.

### 1.8.6 Deb - Nexus 7 (2013 GSM)

This should probably **also work for the Wifi edition (flo)**.

#### Status

#### Halium

**Halium 7.1 is working for this device** thanks to doniks. There is a detailed log by doniks in the ubports forums and a halium/projectmanagement issue. The sources are linked in the forum post.

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS 14.1 info page | android_device_asus_flo | android_kernel_google_msm based on v3.4.0 | ? | ? |

#### Kernel & Hardware

#### Mainline Kernel (4.13rc4 as of writing)

Most of the drivers is already mainlined. Check John Stultzes kernel below for what is missing.

#### The "John Stultz" almost mainline Kernel (4.11 latest)

John Stultz from Linaro has been working some time on getting everything for the Nexus 7 mainline and the most promising candidate to try on this device should be one of his kernel branches. The 4.10 branch is just 23 commits different from mainline! See here a post from John on the mainlining effort.

#### LOS 14.1 Android Kernel (3.4.0)

Unclear whether security bug fixes are implemented into this kernel (long term 3.4 kernel @ kernel.org has version 3.4.113).

#### Canonical's Ubuntu Touch Kernel (3.4.0 based)

This is listed because it used the kernel backports to integrate newer drivers in the orginal vendor kernel which might be usefull in some cases. It can be found here.

#### Device Specifics

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database. For now check the LOS device page linked in the table above.

### 1.8.7 Flo - Nexus 7 (2013 Wifi only)

This is not being worked on right now, however it's the most mainline phone besides the N900 AFAIK.

#### Status

#### Halium

There are currently no effort to get Hallium to work on Flo directly although there is work being done to port it to the Nexus 7 (2013 LTE) aka deb by user doniks. There seem to be cases were images work on both phones which should be the case since deb is flo with additional mobile network capabilites.

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS 14.1 info page | android_device_asus_flo | android_kernel_google_msm based on v3.4.0 | ? | ? |

#### Kernel & Hardware

#### Mainline Kernel (4.13rc4 as of writing)

Most of the drivers is already mainlined. Check John Stultzes kernel below for what is missing.

#### The "John Stultz" almost mainline Kernel (4.11 latest)

John Stultz from Linaro has been working some time on getting everything for the Nexus 7 mainline and the most promising candidate to try on this device should be one of his kernel branches. The 4.10 branch is just 23 commits different from mainline!

#### LOS 14.1 Android Kernel (3.4.0)

Unclear whether security bug fixes are implemented into this kernel (long term 3.4 kernel @ kernel.org has version 3.4.113).

#### Canonical's Ubuntu Touch Kernel (3.4.0 based)

This is listed because it used the kernel backports to integrate newer drivers in the orginal vendor kernel which might be usefull in some cases. It can be found here.

#### Device Specifics

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database. For now check the LOS device page linked in the table above.

### 1.8.8 FP2 - Fairphone 2

The Fairphone 2 is the smartphone made with highest priority for fair production worldwide (AFAIK) and extremely easy to repair (compared to other phones), This makes this a great candidate for a reference device ;-)

#### Status

#### Halium

There are reports of Halium running on the device here, however the locations of the sources are unknown to the author at the moment.

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Ubports 15.04 (Android 5.1 base) device page | android_device_fairphone_fp2 | android_kernel_fairphone_fp2 based on v3.4.0 | ? | ? |

Also there is an official page by the manufacturer dedicated to list more.

#### Kernel & Hardware

#### Mainline (4.13rc4 as of writing)

There is no device tree source inside the mainline kernel specific to the FP2. However, the underlying SoC is present (MSM8974).

#### UBports kernel (3.4.0 base)

This is probably taken from the Fairphone sources directly and patched with the Ubuntu Touch specific stuff, need to check this further to be sure...

#### Device Specifics

#### Guides

This should be populated with guides how to get into different boot modes and similar.

#### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

#### Usefull Ressources

If anything might be usefull but didn't fit above you can just throw in some links here.

### 1.8.9 Hammerhead - Nexus 5

This is one of our **reference devices** and is besides the Nexus 7 2013 (flo) one of the best mainlined smartphones out there.

#### Status

#### Halium

Halium is **working on this device as version 5.1**. You can find the kernel at Halium/android_kernel_lge_hammerhead and the device tree at Halium/android_device_lge_hammerhead.

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS 14.1 device page | LineageOS/android_device_lge_hammerhead | LineageOS/android_kernel_lge_hammerhead based on v3.4.0 | ? | ? |
| UBports 15.04 (Android 5.1 base) device page | UBports/android_device_lge_hammerhead | UBports/android_kernel_lge_hammerhead based on v3.4.0 | ? | ? |

#### Kernel & Hardware

#### Mainline (4.13rc4 as of writing)

Main **device tree source (dts) is available** at qcom-msm8974-lge-nexus5-hammerhead.dts, inside the latest mainline kernel but how far the functionality goes exactly is unknwon. The general support for Nexus 5 landed with 4.9 according to phoronix.

#### Cyanogemod based kernels (LOS & UBP)

Both **UBports(UBP) and LOS share a common base** which is Cyanogenmod. However the Ubports kernel received some patches from the **backports project** (backports on github and the official wiki) whereas the LOS kernel has a few more manual commits.

#### Device Specifics

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database. For now check the LOS device page linked in the table above.

### 1.8.10 krillin - BQ Aquaris E4.5

This device is besides having a smaller display almost identical to the Aquaris E5 so maybe images are interchangeable.

#### Status

#### Halium

There is no one working on bringing Halium to this device AFAIK.

#### Distributions

The following entries are just a placeholder, exactly as this sentence.

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Ubports 15.04 (Android 5.1 base) device page | ? | ? | ? | ? |
| Unofficial LineageOS 13 device page | Pablito2020/android_device_bq_krillin | Pablito2020/android_kernel_bq_krillin based on 3.10.49 | ? | some problems with SMS |

#### Kernel & Hardware

#### Mainline (4.13rc4 as of writing)

There is no Mediatek MT-6582 device source tree present inside the mainline kernel yet.

#### Canonical's kernel (3.4.67 based)

The kernel from canonical especially saw some backporting. AFAIK with this kernel every feature except radio worked.

#### Pablito2020 inofficial LOS kernel (3.10.49 based?)

This kernel (linked in table) seems to be more recent and is said to work completely with a CM13 base.

### Device Specifics

### Guides

### Entering Fastboot & Recovery Mode

For BQ Aquaris E4, E4.5, E5 and E6, assuming *device is off*:

Hold *Power* button + *Volume Up* button until menu pops up

Boot Mode Selection: Navigate with *Volume Up* and select with *Volume Down*

Entering Recovery: Press *Volume Up* to access menu Navigate with *Volume* buttons and select with *Power* button.

### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

### Usefull Ressources

If anything might be usefull but didn't fit above you can just throw in some links here.

## 1.8.11 land - Xiaomi Redmi 3S/Prime

land - Xiaomi Redmi 3S/Prime is a entry-level phone with aggresive development on XDA Forums. The Prime variant has a fingerprint sensor and 3Gb RAM

### Status

### Halium

Halium 7.1 is **WIP**. Ported by rupansh and sridhardv.

### Distributions

| Distribution | Device Specific Files | Device Common Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|---|
| Unofficial LineageOS 14.1 Tree by hyper team | HyperTeam/android_device_xiaomi_land | HyperTeam/android_device_xiaomi_msm8937-common | HyperTeam/android_kernel_xiaomi_msm8937 based on v3.18.31 | see device page | see device page |
| land on Halium projectmanagement | rupansh/android_device_xiaomi_land-1 | rupansh/android_device_xiaomi_msm8937-common | rupansh/android_kernel_xiaomi_msm8937-1 based on v3.18.31 | see projectmanagement issue | see projectmanagement issue |

### Kernel & Hardware

### CAF based

The kernel is based off CAF's kernel (Though it has some LineageOS specific patches)

### Device Specifics

### Guides

Unlocking the bootloader WILL void the warranty for this device.

To unlock the bootloader, you need to get the permission from Xiaomi.

Press Volume Down + Power Button to enter fastboot mode.

Press Volume Up + Volume Down + Power button to enter recovery mode.

### Developer Info

Unlocking bootloader is cancerous for Xiaomi devices.

### Usefull Resources

- TWRP for Xiaomi Redmi 3S/Prime

## 1.8.12 mako - Nexus 4

As all Nexii rather good support of sources.

### Status

### Halium

Work in progress.

Sources:

- Kernel
- Device
- Vendor

What works:

- Halium boots with reference rootfs
- LED
- Vibrator
- Wifi
- Buttons
- Graphics
- Battery

What doesn't work (or isn't tested yet):

- Audio
- Camera

### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Mako on LineageOS Wiki | LineageOS/android_device_lge_mako | LineageOS/android_kernel_lge_mako based on v3.4.0 | ? | ? |
| Ubports 15.04 (Android 5.1 base) device page | ? | ? | ? | ? |

### Kernel & Hardware

### Mainline (4.13rc4 as of writing)

No exact device tree source (dts) for mako so far. But a general dts for the underlying SoC (APQ8064) is available and also used for other specific dts'ses (e.g.) Nexus 7)

### LineageOS kernel (3.4.0 based)

TODO

#### Canonical's Ubuntu Touch kernel (3.4.0 based)

Available here.

### Device Specifics

### Guides

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database.

### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

### Usefull Ressources

If anything might be usefull but didn't fit above you can just throw in some links here.

## 1.8.13 Nicki - Sony Xperia M

Xperia M is very weak, especially by today's standards. But it runs Halium. Somehow.

### Status

### Halium

There is an ongoing port of Halium for Nicki that Konrad Dybcio is working on. Halium-7.1 branch builds with LOS 14.1 sources. Halium rootfs can be flashed using halium-install by JBB, yet /data partition is too small (2 gigs) to fit a plasma mobile rootfs. This can be worked around by changing IMAGE_SIZE=2G to for example IMAGE_SIZE=1.7G for pm in halium-install/functions/distributions.sh.

### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS | android_device_sony_nicki | android_kernel_sony_msm8x27 based on v3.4.0 | Everything(afaik) | Nothing |
| Halium Reference Rootfs | ewen-tual/halium_device_sony_nicki | ewen-tual/halium_kernel_sony_msm8x27 based on v3.4.0 | See projectmanagement issue | Everything else |
| Ubports | Not yet | Not yet | Nothing, even putting rootfs on the device (not enough space) | Everything |

#### Kernel & Hardware

#### Mainline

Not present

#### Cyanogemod based kernels (LOS & UBP)

Kernel version is v3.4.0.

#### Device Specifics

#### Guides

Special boot modes

Fastboot: With the device powered off, while holding Volume Up, connect the USB cable to the computer. The LED should turn blue.

Flashmode: With the device powered off, while holding Volume Down, connect the USB cable to the computer. The LED should turn green. Then you can go back to the stock rom using Androxyde's Flashtool. Last sure to be working with nicki version of Flashtool was 0.9.18.6, newer versions could fail.

Recovery (ONLY on android custom kernels): On boot, press Volume Down when the LEDs start lighting up. This only works with a working boot image. If there is none, you can flash twrp to boot partition first and use it to reboot into recovery.

Second Recovery: To boot to the FOTA Recovery follow recovery instruction, but press Volume Up instead.

#### Developer Info

With hybris-boot only way to access TWRP is to flash it as boot.

#### Useful Resources

- halium-install from JBB

### 1.8.14 oneplus3 - OnePlus 3(T)

The OnePlus 3 (also abbreviated as OP3) is a smartphone produced by OnePlus. It was revealed on June 14, 2016. The OnePlus 3 is the first OnePlus device to not be part of the invite system, which OnePlus had used for its last three devices to regulate flow with inadequate manufacturing for the inevitable high demand.

The OnePlus 3T (also abbreviated as OP3T) is a smartphone made by OnePlus. It is the successor to the OnePlus 3 and was revealed on 15 November 2016. It is an incremental update to the company's flagship phone being released only 6 months later.

### Status

### Halium

Halium is **working on this device as version 7.1**. Ported by Nelly Simkova, Ivan Semkin and Marius Gripsgard

### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| oneplus3 on LineageOS Wiki | LineageOS/android_device_oneplus_oneplus3 | LineageOS/android_kernel_oneplus_msm8996 based on v3.18.31 | see device page | see device page |
| oneplus3 on Halium projectmanagement | Halium/android_device_oneplus_oneplus3 | Halium/android_kernel_oneplus_msm8996 based on v3.18.31 | see device page | see device page |

### Kernel & Hardware

### Cyanogemod based kernels (LOS & UBP)

Both **Halium and LOS share a common base** which is Cyanogenmod. However, the Halium kernel was patched with Halium-specific edits.

### Device Specifics

### Guides

Unlocking the bootloader doesn't void the warranty.

To unlock the bootloader enable `OEM Unlock` setting in Developer Options of your ROM and run `fastboot oem unlock` command while in download mode.

In order to launch download mode, with the device powered off, hold `Volume Up` + `Power`.

In order to launch recovery, with the device powered off, hold `Volume Down` + `Power`.

Also, be sure to check the LineageOS device info for oneplus3

### Useful Resources

- Repo Manifest
- TWRP for OnePlus 3(T)

### 1.8.15 PME (Perfume) - HTC 10

In 2015, HTC released the HTC One M9 smartphone, which was praised for its design, but criticized for being too similar to its predecessor, the HTC One (M8). To address the shortcomings of the M9, HTC released another new phone that year with a new design and hardware: the HTC One A9. The design of the HTC 10 is somewhat of a mix of the M9 and the A9

#### Status

#### Halium

Halium is **working on this device as version 7.1**. Ported by Ivan Semkin and Marius Gripsgard

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| PME on LineageOS Wiki | LineageOSandroid_device_htc_pme | LineageOSandroid_kernel_htc_msm8996 based on v3.18.31 | see device page | see device page |
| PME on Halium projectmanagement | Halium/android_device_htc_pme | Halium/android_kernel_htc_msm8996 based on v3.18.31 | see device page | see device page |

#### Kernel & Hardware

#### Cyanogemod based kernels (LOS & UBP)

Both **Halium and LOS share a common base** which is Cyanogenmod. However, the Halium kernel was patched with Halium-specific edits.

#### Device Specifics

#### Guides

HTC allowed users to officially unlock phones without S-Off. Links provided below

This device has 2 modes for fastboot: `bootloader` (white backgrond) and `download` (dark background). You have to use `download` mode in order to flash/wipe anything. AFAIK bootloader mode is only used to check its status and for emergency purposes.

In order to launch recovery, you have to boot into `download` mode first (Power + Volume Down), then select `reboot to bootloader`, and only then `boot to recovery mode`.

Also, be sure to check the LineageOS device info for PME

#### Developer Info

TWRP's BusyBox for the device is partially broken. Thus stock rootstock might fail to install Halium properly. I suggest using JBB's alternative installation script.

#### Useful Resources

- Repo Manifest
- Official HTC Unlock guide
- TWRP for HTC 10

### 1.8.16 rolex - Xiaomi Redmi 4a

A Snapdragon 425 (MSM8917) budget phone from Xiaomi released in 2016 that is great on its own. Comes with MIUI 8 (Marshmallow), upgradeable to MIUI 9 (Nougat).

It has 2GB LPDDR3 RAM, 16GB/32GB eMMC storage.

An updated version of this device Redmi 5a (riva) was released in 2017. Both are basically the same, and share device trees and kernels. So anything that applies to 4a should as well apply to 5a

#### Status

#### Halium

Halium 7.1 is **WIP**. Ported by echosalik

#### Distributions

| Distribution | Device Specific Files | Device Common Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|---|
| Unofficial LineageOS 14.1 Tree by LineageOS-R4A | LineageOS-R4A/android_device_xiaomi_rolex | NA | LineageOS-R4A/android_kernel_xiaomi_msm8917 based on v3.18.31 | see device page | see device page |
| rolex on Halium projectmanagement | echosalik/android_device_xiaomi_rolex | NA | echosalik/android_kernel_xiaomi_msm8917 based on v3.18.31 | see projectmanagement issue | see projectmanagement issue |

#### Kernel & Hardware

The kernel is based off LineageOS-R4A's kernel (though it has some halium specific patches).

### Device Specifics

### Guides

Unlocking the bootloader WILL void the warranty for this device.

To unlock the bootloader, you need to get the permission from Xiaomi. Takes around a week or two. Offical Guide.

Press Volume Down + Power Button to enter fastboot mode.

Press Volume Up + Power button to enter recovery mode.

### Useful Resources

TWRP XDA Link

## 1.8.17 T00F/T00J - Asus Zenfone 5

Here only one or two sentences. What makes the device special? Is it a reference device?

### Status

### Halium

Halium 7.1 is being worked on by Ilya Bizyaev and as of august 2017, he's got a **working x86 rootfs and LXC container** but test seem to fail. Here are Device files and kernel. The local manifest has the following form

```
<manifest>
  <remote name="zf" fetch="http://github.com/zenfone-legacy" revision="refs/heads/cm-
↪14.1"/>
  <project name="Halium/android_device_asus_T00F" path="device/asus/T00F" remote="hal
↪"/>
  <project name="Halium/android_kernel_asus_T00F" path="kernel/asus/T00F" remote="hal
↪"/>
  <project name="proprietary_vendor_asus" path="vendor/asus" remote="zf"/>
  <project name="android_external_stlport" path="external/stlport" remote="los"/></
↪manifest>
```

There is an issue in halium/projectmanagement for the T00F.

### Distributions

The following entries are just a placeholder, exactly as this sentence.

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Hammerhead on LineageOS Wiki <https://wiki.lineageos.org/devices/hammerhead>`_ | android_device_placeholder | android_kernel_placeholder based on vX.Y.Z | ? | ? |

#### Kernel & Hardware

#### Mainline (version TODO)

TODO

#### Cyanogemod based kernels (LOS & UBP)

If other kernels exist, just make up headlines with a name that describe the origin (Cyanogemod, LineageOS, UBports, Canonical, ...) and which is the underlying mainline version. Afterwards describe what's been improved or altered and if possible why or what is still missing.

#### Device Specifics

#### Guides

This should be populated with guides how to get into different boot modes and similar. For this device this can be pulled from the LOS device database.

#### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

#### Usefull Ressources

If anything might be usefull but didn't fit above you can just throw in some links here.

### 1.8.18 Titan - Moto G 2014

Here only one or two sentences. What makes the device special? Is it a reference device?

#### Status

#### Halium

A port to **Halium 7.1 is being worked on** by Walid Hammami. The according Halium/projectmanagement issue can be found here.

#### Distributions

The following entries are just a placeholder, exactly as this sentence.

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS placeholder device page placeholder | android_device_placeholder | android_kernel_placeholder based on vX.Y.Z | ? | ? |
| Ubports placeholder device page placeholder | android_device_placeholder | android_kernel_placeholder based on vX.Y.Z | ? | ? |

#### Kernel & Hardware

#### Mainline (vX.Y.Z as of writing)

Write whether something that is needed for the device is mainline already (switch the version in the heading for what's recent when you write this). This means **device tree source files (.dts) as well as single drivers** (for example only the wifi driver).

#### Cyanogemod based kernels (LOS & UBP)

If other kernels exist, just make up headlines with a name that describe the origin (Cyanogemod, LineageOS, UBports, Canonical, ...) and which is the underlying mainline version. Afterwards describe what's been improved or altered and if possible why or what is still missing.

#### Device Specifics

#### Guides

This should be populated with guides how to get into different boot modes and similar. Maybe this can be pulled from the LOS device database.

#### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

#### Usefull Ressources

If anything might be usefull but didn't fit above you can just throw in some links here.

### 1.8.19 Vegeta / Vegetahd - BQ Aquaris E5 / BQ Aquaris E5HD

The family of Aquaris E5 devices by BQ are all called vegeta + a special suffix (there is an E5 HD so it's vegetahd for example). However, the hardware is "the same" only in the versions E5 (vegeta) and E5 HD (vegetahd). The newer E5 4G (vegetalte) <devices/vegetalte> which has an LOS 14.1 image is a completely different System and does not share the same hardware. However, the *BQ Aquaris E4.5 (krillin)* is basically the same and maybe it's images will work with minor changes.

### Status

### Halium

There has been no effort to get Halium onto this device so far.

### Distributions

The following entries are just a placeholder, exactly as this sentence.

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| UBports 15.04 (Android 5.1 base) device page | ? | ? | ? | ? |

### Kernel & Hardware

### Mainline (4.13rc4 as of writing)

There is a very rudimentary device tree source file inside kernel 4.13rc4 at dts/mt6589-aquaris5.dts. This still needs a lot of work.

### Canonical's Ubuntu Touch kernel (3.4.67 based)

This kernel can be found at bq/aquaris-e5

### Device Specifics

### Guides

### Entering Fastboot & Recovery Mode

For BQ Aquaris E4, E4.5, E5 and E6, assuming device is off :

> Hold *Power* button + *Volume Up* button until menu pops up
>
> Boot Mode Selection: Navigate with *Volume Up* and select with *Volume Down*
>
> Entering Recovery: Press *Volume Up* to access menu Navigate with *Volume* buttons and select with *Power* button.

### Developer Info

Some devices show strange behaviour of some kind, try to find this (for example in the xda-developers forum) and document it

#### Usefull Ressources

If anything might be usefull but didn't fit above you can just throw in some links here.

### 1.8.20 Xiaomi Redmi 2/Prime (wt88047)

Xiaomi Redmi 2 smartphone was launched in January 2015. The phone comes with a 4.70-inch touchscreen display with a resolution of 720 pixels by 1280 pixels at a PPI of 312 pixels per inch.

#### Status

#### Halium

A port for Halium 7.1 is being worked on by Abhishek Mudgal

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| WT88047 on LineageOS Wiki | LineageOSandroid_device_wingtech_wt88047 | LineageOSandroid_kernel_wingtech_msm8916 based on v3.10.49 | see device page | see device page |

#### Kernel & Hardware

#### Mainline (vX.Y.Z as of writing)

Write whether something that is needed for the device is mainline already (switch the version in the heading for what's recent when you write this). This means **device tree source files (.dts) as well as single drivers** (for example only the wifi driver).

#### Cyanogemod based kernels (LOS & UBP)

Both **Halium and LOS share a common base** which is Cyanogenmod. However, the Halium kernel was pacthed with Halium-specific edits. Currently only kernel flags in `lineageos_wt88047_defconfig` were altered.

#### Device Specifics

#### Guides

Bootloader for wt88047 comes pre unlocked.

Flash hybris-boot as a boot image from fastboot.

In order to launch recovery, do press "Power + Volume Down + Volume UP"

Also, be sure to check the LineageOS device info for WT88047

#### Developer Info

This device is mostly open and has no weird behaviour

#### Useful Resources

- Repo Manifest
- TWRP for Xiaomi Redmi2/Prime

### 1.8.21 Yuga - Sony Xperia Z

There exists basic mainline kernel support added by Sony. This possibly could become a reference device.

#### Status

#### Halium

There is a **working Halium 7.1** port made by LNJ and here is the according Halium project management issue.

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Yuga on LineageOS Wiki | LineageOS/android_device_sony_yuga | LineageOS/android_kernel_sony_apq8064 based on v3.4.113 | everything | |

#### Kernel & Hardware

#### Mainline (v4.15-rc9 as of writing)

The kernel will start and there is already UART and USB debugging implemented for this device. There also has been worked on riva, which is used for connectivity like WiFi, Bluetooth and FM, but there is no documentation of what really works.

#### LineageOS kernel (v3.4.113)

This is just the original Sony kernel with some security fixes, updates and back ports. Unfortunately the LTS has ended, so 3.4.113 is the last version.

### Device Specifics

### Guides

You can always get into fastboot by holding the power button and the volume up button until the device vibrates three times, then only press the volume up button and you'll get into fastboot. If you use a LineageOS/CyanogenMod boot image you can also press volume up or down, when the LED turns pink on startup.

I always had to flash TWRP to the boot partition to get into the recovery again. Currently I don't know if there is any way to get into the recovery in a simpler way.

### Developer Info

There's no strange behaviour, except the way to get into fastboot or recovery.

### Useful Resources

TWRP for yuga (only up to v3.0.2)

## 1.8.22 Taoshan - Sony Xperia L

The device is quite weak to run Halium. But it should be possible.

### Status

### Halium

A port for Halium 7.1 is being worked on by Jonatan Hatakeyama Zeidler. Halium 7.1 builds, but does not install using halium-install due to TWRP BusyBox issues. Using halium-install from JBB the halium reference rootfs can be installed, but it does not seem to start.

### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| LineageOS | LineageOS/android_device_sony_taoshan | LineageOS/android_kernel_asus_msm8916 based on v3.4.0 | Everything | Nothing |
| Halium Reference Rootfs | jonnius/android_device_sony_taoshan | jonnius/android_kernel_asus_msm8916 based on v3.4.0 | Installation | Does not boot |
| Plasma Mobile | jonnius/android_device_sony_taoshan | jonnius/android_kernel_asus_msm8916 based on v3.4.0 | Building images | Installation (/data is to small) |
| UBPorts | None | None | Porting not yet started | Everything |

### Kernel & Hardware

### Mainline

I don't know if something that is needed for the device is mainline already.

### Cyanogemod based kernels (LOS & UBP)

Kernel version is 3.4.0.

### Device Specifics

### Guides

Special boot modes

- Fastboot: With the device powered off, while holding Volume Up, connect the USB cable to the computer. The LED should turn blue.
- Recovery: On boot, press Volume Down when the LEDs start lighting up. This only works with a working boot image. If there is none, you can flash twrp to boot partition first and use it to reboot into recovery.
- Second Recovery: To boot the FOTA Recovery, press Volume Up instead. This seems to be equal to Recovery.

### Developer Info

After flashing hybris-boot, recovery can not be accessed anymore. Therefore follow these steps to install halium rootfs:

- Enter Fastboot
- Flash TWRP image to recovery and boot partition
- Reboot into TWRP on boot
- From TWRP reboot into Recovery
- In TWRP mount data partition
- Clone halium-install from JBB (the official one does not work)
- Edit functions/core.sh and change rootfs size to 1G (data partition has only 1.6 GB)
- Run halium-install with halium rootfs image and system.img
- Enter Fastboot
- Flash hybris-boot image to boot partition using fastboot

### Usefull Resources

- halium-install from JBB

### 1.8.23 Z00L - Zenfone 2 Laser

The Zenfone 2 Laser is a group of mid-range Android smartphones, released in August 2015, with a number of different region-specific configurations. The Zenfone 2 Laser comes with either a 5", 5.5" or 6" display, with either 720p or 1080p resolutions. Chipsets range from a 1.2GHz quad-core Snapdragon 410, to a 1.7GHz octa-core Snapdragon 616, with each device having either 2GB or 3GB of RAM and 8GB, 16GB or 32GB internal storage. All devices are capable of holding a microSD card up to 128GB and have dual-SIM capability. Cameras comes in either 8MP or 13MP and batteries range from 2400mAh to 3000mAh.

#### Status

#### Halium

A port for Halium 7.1 is being worked on by Aayush Gupta

#### Distributions

| Distribution | Device Specific Files | Kernel | What works | What doesn't work |
|---|---|---|---|---|
| Z00L on LineageOS Wiki | LineageOS/android_device_asus_Z00L | LineageOS/android_kernel_asus_msm8916 based on v3.10.108 | see device page | see device page |
| Z00L on Halium projectmanagement | Halium/android_device_asus_Z00L | Halium/android_kernel_asus_msm8916 based on v3.10.108 | see device page | see device page |

#### Kernel & Hardware

#### Mainline (v3.10.108 as of writing)

Everything required is already patched in kernel source and is fully upstreamed. Patches from CAF and Linux are also merged already.

#### Cyanogemod based kernels (LOS & UBP)

Both **Halium and LOS share a common base** which is Cyanogenmod. However, the Halium kernel was pacthed with Halium-specific edits. Currently some kernel flags and additional patches were used.

#### Device Specifics

#### Guides

Special boot modes

> Recovery: With the device powered off, hold Volume Up + Power. On the next screen (fastboot mode) use Volume Down to scroll to "RECOVERY" and then press Power to select.
>
> Download: With the device powered off, hold Volume Up + Power.

#### Developer Info

This device is competely open and has no weird behaviour

#### Useful Resources

- TWRP for Z00L

### 1.8.24 How to document

#### Purpose

This table is meant for 2 purposes:

- 1. give porters a **quick status reference** what has already been done in regard of Halium to a specific device or what the starting conditions are
- 2. make hardware visible for **kernel team** to see there may be a possibility of **a common drivers base** for multiple devices

#### Table Format

Please use the following where the formatting is not obvious:

1. First Row 'Device' is Manufacturer + Device Name and **links to the devices sub page**
2. Second Row 'Codename' is the official Codename as stated inside the Android/Ubuntu Touch/Lineage sources
3. Third Row 'Halium status' is a very short description like 'working fully' or 'boot loop' with the base version attached so e.g. with halium 5.1 base: 'working fully - 5.1'
4. All following Hardware Rows **should link to a detail page about every chip**
5. If you **only have the labels from chips available** just set them inside 'label' perhaps someone else knows more details (most common for cameras, see Nexus 4 example below)
6. **Kernel Version Please fill this one only when there is an existing Halium image!**

#### Getting the Info

- Search **Wikipedia** for your device, a good starting points are the articles that compare smart phones or tablets
- Search the xda-developers forum
- Connect to your device via adb (assuming you have some kind of Android on your phone/tablet) and search for infos using getprop or similar
- Search for **teardowns** and scan the images for readable chips and try to find out what each one does (you may of course do a teardown on your own) good starting points for this are iFixit or Youtube Videos that show how to repair/tear down a device
- Search the LineageOS device database for your devices codename.
- Please **write down your sources somewhere and link to them below the table or in the device's sub page**

### 1.8.25 Devices

| Device | Codename | Halium status | Kernel | SoC | CPU | GPU | RAM | Storage | Connectivity | Camera Front | Camera Back | Battery | Sound | Touchscreen | Display | Navigation | Sensors | Other |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Oneplus 5 | cheeseburger | Halium 7.1 WIP | v4.4.78 | Qualcomm Snapdragon 835 | Octa-core CPU | Qualcomm Adreno 540 | 8 GB LPDDR4 | 64 GB UFS (MMC flash + controller) | Qualcomm WCN3990 (WiFi), NXP 80T71 (NFC) | Sony IMX371 | Dual Camera with Sony IMX398 (wide angle) + Sony Exmor IMX350 ("telephoto") | Battery 3300 mAh | Qualcomm WCD9341 (audio codec) | Touchscreen | Display | Navigation | Sensors | Other |
| *Samsung Galaxy S8+* | dreamlte | Halium 7.1 WIP | v4.4.78 | Samsung Exynos 9 Octa 8895 | 4x 2.3 GHz Exynos M2 Mongoose, 4x 1.7 GHz ARM Cortex-A53 | Mali-G71 MP20 | Samsung K3UH6H60AM-NGCJ 4 GB LPDDR4 | Toshiba THGBMFG8C4LBAIR 64 GB UFS (MMC flash + controller) | Qualcomm WCN3990 (RF transceiver), Murata KM7118064 (WiFi), NXP 80T71 (NFC) | Sony Exmor RS (8MP) | Sony Exmor RS (12MP) | Battery 3500 mAh 3.85V, 13.48 Wh | Qualcomm WCD9341 (audio codec) | Touchscreen ?? | Super AMOLED (1440x2960) | GPS, GLONASS, BDS, GALILEO | Proximity, Light, Accelerometer, Compass, Gyroscope, Barometer, Fingerprint, Hall, Heart rate | Skyworks 78160-11 (?), Avago AFEM-9066 (?), Avago AFEM-9053 (?), Silicon Mitus SM5720 (Power Management), Qualcomm PM8998 (Power Management) |
| Samsung Galaxy S8 | dreamlte | Not working so far | v4.4.78 | Samsung Exynos 9 Octa 8895 | 4x 2.3 GHz Exynos M2 Mongoose, 4x | Mali-G71 MP20 | Samsung K3UH6H60AM-NGCJ 4 GB LPDDR4 | Toshiba THGBMFG8C4LBAIR 64 GB UFS (MMC flash | Qualcomm WCN3990 (RF transceiver), Murata KM6D28040 | Sony Exmor RS (8MP) | Sony Exmor RS (12MP) | Battery 3000 mAh 3.85V, 11.55 Wh | Qualcomm WCD9341 (audio codec) | Touchscreen ?? | Super AMOLED (1440x2960) | GPS, GLONASS, BDS, GALILEO | Proximity, Light, Accelerometer, Compass, | Skyworks 78160-11 (?), Avago AFEM-9066 |

### Sources

Since there are no sub pages yet, gathering links for the examples here:

### OnePlus 5

- Oneplus 5 iFixit teardown
- Oneplus 5 Wikipedia

### Samsung Galaxy S8+

- Samsung Galaxy S8+ iFixit teardown
- Samsung Galaxy S8+ GSMArena
- Samsung Galaxy S8+ DeviceSpecifications
- Samsung Galaxy S8(+) Website

### Samsung Galaxy S8

- Samsung Galaxy S8 iFixit teardown
- Samsung Galaxy S8 GSMArena
- Samsung Galaxy S8 DeviceSpecifications
- Samsung Galaxy S8(+) Website

### Shift 5.1

- Shift 5.1 iFixit teardown

### Wiko Pupl 4G

- Pupl 4G iFixit teardown

### Nexus 5

- Nexus 5 iFixit teardown
- Nexus 5 Wikipedia
- Nexus 5 Slideshare on Snapdragon 800

### Nexus 5X

- Nexus 5X iFixit teardown
- Nexus 5X Wikipedia

### Oneplus One

- Oneplus One iFixit teardown
- Oneplus One Wikipedia

### Nexus 4

- Nexus 4 iFixit teardown
- Nexus 4 Wikipedia
- Nexus 4 Kernel @ launchpad

### Fairphone 2

- Fairphone 2 iFixit teardown
- Fairphone 2 UBP dev info page
- Fairphone 2 Wikipedia
- Fairphone 2 Kernel Makefile @ github/UBP

### Nexus 7 (2012)

- Nexus 7 2012 iFixit teardown
- Nexus 7 2012 Wikipedia

### Nexus 7 (2013) Wifi-only = flo

- Nexus 7 2013 iFixit teardown
- Nexus 7 2013 Wikipedia

### BQ Aquaris E4.5

- Aquaris E4.5 Wikipedia

### HTC 10

- HTC 10 Wikipedia
- HTC 10 GSMArena
- HTC 10 DeviceSpecifications
- HTC 10 iFixit
- HTC 10 LOS
- HTC 10 Website

### Asus Zenfone 2 Laser (Z00L)

- Asus Zenfone 2 Laser Z00L LOS

### Xiaomi Redmi2/Prime (wt88047)

- Xiaomi Redmi 2/Prime GSM-Arena
- Xiaomi Redmi 2/Prime Fonearena
- Wingtech Wt88047 LOS

### OnePlus 5

- OnePlus 3 Wikipedia
- OnePlus 3T Wikipedia
- OnePlus 3 GSMArena
- OnePlus 3T GSMArena
- OnePlus 3 DeviceSpecifications
- OnePlus 3T DeviceSpecifications
- OnePlus 3 iFixit teardown
- OnePlus 3(T) LOS
- OnePlus 3 Website
- OnePlus 3T Website

### Xiaomi Redmi 3s/Prime

- Redmi 3s Website

### Xiaomi Redmi 4A

- Redmi 4A Website
- Redmi 4A GSMArena

## 1.9 Hardware and kernel enablement

This page is meant as an overview of

- **What Hardware may be inside a smart phone**
- **Which of those parts need a kernel driver** = probably all
- **What is eachs components current status?** = mainline(since)/wip/blob

**Please be aware that mainlining is A PROCESS** so sometimes "mainline" doesn't mean the same for two chips. However we will use the term in this overview as in "the chip is being mainlined beginning with kernel v.xx) and a detailed overview shall be given on a sub page or an external reference.

### 1.9.1 How to use this Overview

If you are trying to document the parts of a new or incomplete documented device the chips should all be listed in the *Device Overview* and on each devices sub page.

Every hardware part should be listed on this page in it's category (connectivity, sensors, ...) and link to a **detailed sub page** with further info if possible **while only open source status should be presented here**.

Also in the tables below, it would be very useful to know with which kernel version certain drivers were mainline so please try to take the time to **document the kernel version of each's components mainlining success**

#### Where to find software status

How to find all hardware related data is documented on the Devices Overview. However, it's not that trivial to find out if there is an open source (or even mainlined) driver of the specific component. Essentially, you need to search the linux kernel and drivers should be inside the folder *drivers/*. **Always keep in mind that abbreviations are used** e.g. qcom instead of Qualcomm.

There are many ways to search for specific drivers. I'll need to tinker a little bit or ask around to find out which way works best. All ways shall be listed below

1. You can for example manually browse the mainline kernel by clicking on 'browse' there, then navigate to the 'tree' section and then click yourself through the drivers folder
2. You can search linux kernel mailing lists like the linux-arm-kernel archives for specific hardware
3. For **Qualcomm SoCs** there is a community driven qualcomm mainlining wiki as is for **Allwinner** @ sunxi mainlining
4. You can also look into kernel changes on kernelnewbies.org
5. Wicket from the Maemo community also has a short list of devices and their status pages.

#### SoC support @ Device Tree Source (dts) folder

Modern ARM devices should have their own **device tree** inside the mainline kernel. For a really good reference check out 'Device Tree for Dummies' by Thomas Petazzoni from Free Electrons. Also from Thomas Petazzoni is the great ARM-soc-checklist (they really produce stunning docs there)

For all ARM based smart phones and tablets this should be inside

```
root/arch/arm/boot/dts/
```

or if your device is using a 64-bit (arm64) processor (e.g. BQ Aquaris M10) all files are sorted by manufacturer inside

```
root/arch/arm64/boot/dts/
```

## 1.9.2 System Components

### System-on-Chips (SoC)

### Device specific dts

**Please only include SoCs that are used inside any smart phone/tablet** and have a smart phone/tablet **.dts* file.

| Manufacturer | SoC | Board | Mainline(since) | file | armhf/arm64 |
|---|---|---|---|---|---|
| Qualcomm | APQ8064 | Asus Nexus 7 | since 4.6 | qcom-apq8064-asus-nexus7-flo.dts | armhf |
| Qualcomm | APQ8064 | Sony Xperia yuga | ? is inside 4.12-rc2 | qcom-apq8064-sony-xperia-yuga.dts | armhf |
| Qualcomm | MSM8974 | LG Nexus 5 | ? inside 4.12-rc2 | qcom-msm8974-lge-nexus5-hammerhead.dts | armhf |
| Qualcomm | MSM8974 | Sony Xperia Honami | ? inside 4.12-rc2 | qcom-msm8974-sony-xperia-honami.dts | armhf |

Here, we directly see what is the benefit of these tables -> The Qualcomm MSM8974 SoC is already mainlined and there are several boards one can use as a reference to e.g. **write a fairphone2.dts** with the purpose to one day mainline it! Also, the APQ8064 SoC is used inside Nexus 4 as well. . .

### General dts

Below are gathered mainlined SoCs and Smart phones/Tablets that use it but do not have a device-specific **.dts* yet.

| Manufacturer | SoC | armhf/arm64 | Smartphones/Tablets |
|---|---|---|---|
| Qualcomm | APQ 8060 | armhf | HP TouchPad, HTC Jetstream, HTC Amaze 4G, HTC Vivid, HTC Raider 4G, Le Pan II, LG Nitro HD, Pantech Element, Samsung Galaxy S II X (SGH-T989D), Samsung Galaxy S II LTE, Samsung Galaxy S II Skyrocket, Samsung Galaxy S Blaze 4G, Samsung Galaxy Tab 7.7 LTE, Samsung SGH-i577 Galaxy Exhilarate, Sony Xperia ion |
| Qualcomm | APQ 8064 | armhf | Asus PadFone 2, HTC Droid DNA, HTC J Butterfly, LG Optimus G, Nexus 4, Oppo Find 5, Pantech Vega No.6, Pantech Vega R3, Sharp Aquos Phone Zeta (SH-02E), Sony Xperia UL, Sony Xperia Z, Sony Xperia ZL, Sony Xperia ZR, Xiaomi MI-2, Panasonic P-02E,ZTE Nubia Z5,ZTE Nubia Z5 Mini |
| Qualcomm | APQ 8074 | armhf | ? |
| Qualcomm | APQ 8084 | armhf | Samsung Galaxy S5 LTE-A (Korea), Samsung Galaxy S5+, LG G3 Cat.6, Samsung Galaxy Note 4 LTE, Samsung Galaxy Note Edge, Amazon Fire HDX 8.9, Motorola Moto MAXX, Motorola Droid Turbo, Nexus 6, Inforce IFC6540, Samsung Galaxy Note 4 Duos |
| Qualcomm | MSM 8660 | armhf | HTC Evo 3D (CDMA), HTC Rezound, LG Connect 4G, LG Optimus LTE LU6200, Pantech Vega Racer, Pantech Sky LTE EX, Pantech Burst, LG Lucid, Samsung Galaxy Note, Xiaomi MI-One (CDMA2000 for China Telecom) |
| Qualcomm | MSM 8960 | armhf | Asus Transformer Pad Infinity (3G/4G version), BlackBerry Z10, BlackBerry Classic HTC Droid Incredible 4G LTE, HTC Evo 4G LTE, HTC One X (North America), HTC One XL, HTC Windows Phone 8X, Samsung Ativ S, LG Mach, Motorola Atrix HD, Motorola Droid Razr M, Motorola Droid Razr HD, Motorola Razr HD, Motorola Droid Razr Maxx HD, Nokia Lumia 820, Nokia Lumia 920, Nokia Lumia 925, Nokia Lumia 1020, Panasonic Eluga Power, Qualcomm Snapdragon S4 Plus MSM8960 MDP/S, Samsung Galaxy S III (select versions), Sharp Aquos Phone sv (SH-10D), Sharp Aquos Phone Zeta (SH-09D), Sony Xperia GX, Sony Xperia TL, Sony Xperia SX, Sony Xperia V, Toshiba Regza Phone (T-02D), ZTE Grand Era LTE, ZTE Grand X LTE,[75] ZTE V96, Huawei Ascend P1 LTE, Kyocera Hydro Elite C6750, Nokia Lumia 928, Nokia Lumia 822, Nokia Lumia 810, Sony Xperia T LTE, LG Lucid 2, LG Optimus F7, LG Optimus F5, LG Spectrum 2, LG Optimus VU II, BlackBerry Q10, Huawei Premia 4G, ZTE Vital, ZTE Avid 4G, ZTE Flash, Dell XPS 10, LG Escape, LG Optimus LTE II, Kyocera Hydro XTRM, Kyocera Torque, BlackBerry Porsche Design, Pantech Discover, Pantech Perception, Pantech Flex, Pantech Vega PTL21, Xolo LT900, Kyocera Torque SKT01 |
| Qualcomm | MSM 8974 | armhf | ? |

Useful sources:

- Wikipedia list of Qualcomm SoCs with devices using them
- Wikipedia list of Tegra SoCs and devices using them

### GPU

There is sadly only a very short list of open source driver projects, mainly Etnaviv which is for VIvante GCxxx embedded GPUs and Freedreno for Ardreno GPUs (there was also once lima for mali GPUs, however it seems more or less abandoned, although just recently there was news that someone from AMD is trying to rewrite it as mesa-lima).

#### Qualcomm's Adreno

- Adreno 320

- Adreno 330
- Adreno 418

### Wireless Connectivity

#### WiFi + (x)

- Broadcom BCM4339 (inside **mainline** 4.11.2 @ /arch/x86/platform/intel-mid/device_libs/platform_bcm43xx.c also perhabs @/drivers/net/wireless/broadcom/b43)
- Qualcomm QCA6174
- Qualcomm WCN3680 (inside **mainline** 4.11.2 @ /drivers/net/wireless/ath/wcn36xx)
- Qualcomm WCN3680B (inside **mainline** 4.11.2 @ /drivers/net/wireless/ath/wcn36xx)
- Murata SS2908001
- AzureWave AW-NH665
- Qualcomm Atheros WCN3660 (inside **mainline** 4.11.2 @ /drivers/net/wireless/ath/wcn36xx)
- Skyworks SKY85709

#### Mobile

- Qualcomm WTR1605L
- Qualcomm WTR3925
- Avago ACPM7800
- Avago ACPM-7251
- Qualcomm MDM9215M

#### NFC

- NXP 65N04
- Broadcom 20793S

### Sound

this is probalby located @ /sound/soc/codecs

- Qualcomm WCD9320
- Qualcomm WCD9330
- Qualcomm WCD9310
- Realtek ALC5642 (not inside 4.11.2 but alc5623)

### Sensors

#### Touch

probably at /drivers/input and than ../mouse or ../touchscreen

- synaptics e.g has a driver in 4.11.2 @ /drivers/input/mouse

#### Orientation & Acceleration

this means gyroscopes, accelerometers, compasses and GPS, probably inside /drivers/iio/

- InvenSense MPU-6515
- Asahi Kasei AK8963
- InvenSense IDG-2020
- ST Microelectronics LSM330DLC
- InvenSense MPU-6050 (inisde **mainline** 4.11.2 @ /include/linux/platform_data_invensense_mpu6050.h & /drivers/iio/imu/inv_mpu6050/)
- Broadcom BCM4751 (general inside **mainline** 4.11.2 @ /arch/mips/bcm47xx/)

### Peripherals

#### Charging & Power Management

- Texas Instruments BQ51013B
- Texas Instruments BQ24192 (not inside 4.11.2 but similar stuff @ /include/linux/power/)
- Qualcomm PM8921
- Qualcomm PM8821
- Qualcomm PM8941 (inside **mainline** 4.11.2 @ /arch/arm/boot/dts/qcom-pm8941.dtsi & /drivers/input/misc/pm8941-pwrkey.c & /drivers/video/backlight/pm8941-wled.c)
- Qualcomm PM8841 (inside **mainline** 4.11.2 @ /arch/arm/boot/dts/qcom-pm8841.dtsi)
- Qualcomm QFE1100
- Qualcomm PMI8994 (inside **mainline** 4.11.2 @ /arch/arm/boot/dts/qcom/pmi8994.dtsi)
- Qualcomm SMB1358

#### Slimport

- Analogix Semiconductors ANX7808 e.g. @ mako kernel 3.4: root/drivers/misc/slimport_anx7808 **mainlined** @ 4.11.2: root/drivers/gpu/drm/bridge/

### 1.9.3 How to test a new kernel

There is some nice documentation by Simon Raffeiner at his blog how to boot a new kernel in Ubuntu Touch (MediaTek devices only!) once for testing.

---

**Todo:** Add info about this on Android/Others

---

## 1.10 How to contribute documentation

**DISCLAIMER: this is still a work in progress and some parts are missing, however this will change soon and you can use it as it is for now**

Okay, we'll assume that you want to help in other ways than donating, bug reporting/testing, translating or organizing/moderating but instead want to document.

Why is this useful? If you have device x at home and want to have a working UBports (Ubuntu Touch)/Plasma Mobile/Halium image but there is no developer out there right now who wants to take on this project you might think *I'll just post a request in the forum, I can't do anything anyway* but **that's wrong**. If you gather information about **software support (Cyanogenmod/Lineage OS/Android images and Kernels available)** what **Hardware is inside exactly**. You have already done the first steps and some dev might think he/she gives it a quick look.

If you now compare the exact hardware with existing, ported devices you **may find that most/some of the hardware is already supported on another device**. If this is the case, you can go a step further and try to find out where exactly in the code these hardware parts are mentioned and if they are also included in the so called 'mainline' linux kernel.

You think this sounds hard and as if you need dev skills? We will try to help you take this steps without any starting knowledge!

For all those who read this but have a little more knowledge about some step/want to use different editors or tools to get to the end: This guide will only show one way and no alternatives at all for better clarity for beginners (for example, you can edit this wiki with many editors but we will recommend typora for now). There will also be nothing about github accounts, pull requests or anything but rather an approach that inexperienced users can do themselves. A and one final thing, some points are simplified to a great extend so don't take anything too literal!

### 1.10.1 Check for existing Halium work

1. **Check the *Device Overview* page to see if your device has already been inserted**
    (a) If it is **inside the list** BUT has no image for download available based upon Halium, see if the entry has all points covered or if info is missing. Also check the device specific sub page and the hardware parts' sub pages if there is some info that has not been inserted yet. Even if all those exist, the info has been gathered, perhaps something has changed since then?
    (b) If it is **NOT in the list**, continue to the next section

### 1.10.2 Download templates and files

1. Getting the current wiki & templates for all sites we want to create

Head over to this documentation's GitHub repository, click on the green button **download or clone** and select *Download ZIP* from the drop down and unpack it in a folder of your choice and move to the folder "supplementary/devices". Inside there are a few files of interest for us:

- **index.rst** here we will insert a quick list of the status of the device and link to a new device specific sub page we will create
- **devicetemplate.rst** this will be our device template, that means we will copy this file, rename it to match our device name and fill the specific info about it

You can open each of these in a multitude of editors but the simple **gedit** which is pre-installed on Ubuntu suffices.

If we will go deeper later on, we might have to create chip-specific sub pages as well and edit the 'Hardware and Kernel Enablement' page but don't worry about this for now.

### 1.10.3 Edit device-overview

1. Open up your browser of choice and head over to the *Device Overview* again. You can use this for reference while editing it locally on your PC and also there is some useful links there that may lead you to the places where you find info about your device without opening it yourself.
2. Also open index.rst in gedit
3. Then, just fill in the voids in the editor. Once finished you can save your work send in your changed index.rst to docu@ubports.com THIS MAIL ADRESS NEEDS TO BE FIXED!
4. Wait until someone merged it into the official wiki

**Note**:Alternatively, you can also create a github account, fork the wiki and create a pull request once you have edited everything you wanted in your branch. How this is achieved will be documented in a future post. But it is not that hard, you can find a quick interactive intro for git here (it will take ~15 minutes)

### 1.10.4 Create device documentation

**Todo:** create reference-device.rst

1. Open up your browser and head over to the *template for device specific pages*. This will again be our reference and may provide some useful links as well.
2. Also open up the **devicetemplate.rst** file with your editor and directly save-as *Your-Device-x.rst* so you always change the correct file and not the template.
3. Now look into the showcase and try to gather as much info about your device in the same style, save your work and submit in the same way as in step 3.

Okay, by now you may have noticed it's not that hard, it's just a lot of work to do this. You may also notice that if info is gathered in a centralized manner, devs may save lot's of time!

### 1.10.5 Check hardware enablement

So this needs a very short intro: Every chip you might imagine that is inside your device needs a driver. This driver needs to be inside the so called *kernel* which is the core of every Android/GNU linux system. There is the **mother of all kernels**, the so called *mainline kernel* which is worked on by thousands of people all over the world.

If all code (= all drivers) needed to run a device is inside this mainline kernel, it will stay there and other people will also take care about not destroying anything when a new version is developed and we can just use every new version that comes along. This kernel changes, sometimes in very radical steps which is why a driver that was once included in a custom kernel, let's say kernel 3.4-mydevice can't just be copied to 4.11 into the same directory. If we stick to

a custom kernel, our little team needs to put all the new, sometimes radical changes into our custom kernel which is much more work on much fewer shoulders in comparison to mainlining our little device specific code.

The biggest issue however is not that the kernel changes but rather **that manufacturers include code that cannot be read or changed by us (legally)** (so called proprietary *blobs*). Which is why we are often stuck at older kernel versions. BUT the **community is often developing alternative free and open source drivers** to replace these blobs. At a certain point, it is possible to 'suddenly' run a device with a new kernel (for example the Nexus 7 2013 is step-by-step mainlined by John Stultz and others). **Our porting devs cannot watch the kernel all the time so we as community can try to check which hardware is supported how well**.

Also, we can link to older, working kernels and where inside of them the specific drivers sit. Some day someone might have the time to take a look and then only needs to look up the paths we provide instead of searching themselves.

So what to do:

1. Open up your browser and head over to the *Hardware and kernel enablement* page.
2. Open up the same page in your editor (**supplementary/hardware-enablement.rst**)
3. Insert missing info and submit your changes
4. If you find a part without its own subpage, you might want to create one. Head to the next step for that.

### 1.10.6 Create pages for undocumented hardware

**Todo:** add component_template.rst

If there is no sub page for a certain hardware component (e.g. a Wifi + Bluetooth chip) you can also create a new hardware component sub page from the template. Just open up the *New-component-template.rst* file, save as *Your-component.rst*, fill with info and submit to the docs team.

## 1.11 Related projects

There are a number of other free software projects to provide a *Linux experience* for mobile devices. Some of these projects are actively working on moving to Halium. All of them are kindred spirits in the quest for Linux on mobile. Not the least, they have valuable resources available at their respective sites about porting to mobile devices.

### 1.11.1 LuneOS

LuneOS is based on Open webOS. It is provided by the WebOS Ports team. The system has it's legacy in projects by LG, HP and Palm. There are ongoing conversations about a cooperation with Halium.

### 1.11.2 Plasma mobile

Plasma-mobile is based on the Plasma shell, developed by the KDE community. Plasma mobile is available on Halium.

### 1.11.3 postmarketOS

PostmarketOS is based on Alpine Linux. It has a selection of multiple phone interfaces. A main difference to Halium is that postmarketOS targets upstream kernels. This brings the advantage of more up to date software. It has the disadvantage that device drivers which are only available as binaries for old kernels can not be used.

### 1.11.4 SailfishOS

Sailfish OS is based on the Mer Project. The user interface is developed by Jolla and not open source. The technical architecture should be largely compatible with Halium.

### 1.11.5 Ubuntu Touch

Ubuntu Touch is based on Unity 8 and is developed by UBports. It is the continuation of the earlier Ubuntu Touch project from Canonical. UBports is actively working on making their future releases available on Halium.

### 1.11.6 Others

There are more projects that share similar goals and/or similar tools and might have valuable resources. A few shout outs: LineageOS, MaruOS, Neo900, Purism Librem 5, Replicant, Tizen.

## 1.12 Planning

### 1.12.1 Linux kernel + Android + Libhybris common basel

#### Introduction

Overall idea is, This stack includes,

- Linux kernel
- Android HAL
- Sensors
- Camera
- RILd
- Libhybris
- Android HAL interfaces like Audioflingerglue and droidmedia
- Build system and scripts
- GPS - AGPS from Mozilla
- Pulseaudio
- Media codecs
- oFono
- Way to distribute updates and images (way to install) (that's debatable)
- Systemd? (Upstart can handle user-level service initialization: See Ubuntu Touch)

This stack doesn't include the,

- Qt
- No Qt would be best, most platforms are very strict on the version they depend on
- Wayland
- KWin hwcomposer platform

- Plasma
- Unity
- Mir
- Lipstick
- Gecko
- Applications
- . . . .

#### Action Points

- See if there are any conflicting requirements and how best to resolve them. For eg. Ubuntu touch needs apparmor patches in kernel, while Sailfish doesn't.
- Decide what the stack will contain and make a short summary of the whole thing.
- Decide how the lxc/container should be setup (img file like ubuntu or in rootfs like sailfish)
- Decide what will be the default method to accrue android, build our own or take vendors and modify that (both would be best here in my opinion)
- What infrastructure we will need for this? What are options?
- Fork https://github.com/mickybart/gnulinux_support / make pull requests?
- Find all kernel features systemd requires, maybe fork mer-kernel-check to automatically check kernel configs
- Fork https://github.com/ubports/ubports-installer for cross platform installations
- Fork ubp-5.1 && ubp-7.0
- Common support for Multirom :) YES PLEASE I don't think Multirom is maintained anymore, but we can fork it can continue it ; There's already a fork with active developers: https://github.com/multirom-dev
- Please let's move away from android recovery update method (it does not fit us) (Boot from sdcard using efidroid at least for testing, so write images to sdcards?)
- Document all the things! YES! We need that badly

### 1.12.2 REVIEW ONCE THEN MOVE UP - bshah

#### What this base consists of? Or what is our "products"?

- AOSP source tree (LineageOS)
- Collection of Device Repos (similar to Cyanogenmod)
- Prebuilt and ready-to-integrate android images
- Reference packaging of libhybris and co. for the distributions / developers
- Reference binary images of this stuff
- Tool to flash the images
- Documentation on how to integrate the AOSP base with Linux system (this is most important.. Basically everything we create needs to be documented from start)
- Common config system?

What I would expect from this is a minimal booting android with libhybris, and all the libhybris tests pass

#### What is the lifecycle of port?

- Porting team decides upon target (or someone suggests it)
- We create android device and vendor tree or "fork" it from LineageOS/CM/AOSP/whatever
- Integrate our changes required for libhybris etc
- CI builds image and publishes them

#### Distributions (Plasma Mobile, UBports, Mer etc)

- Distributions picks up this new packaging if they are using our reference packaging for userspace (Linux) parts, otherwise they build their own packages.
- Along with rootfs generated from side of Distribution, they use the android IMG from the port lifecycle. (kernel depends on splitting / not splitting android and kernel)

### 1.12.3 The Stack - proposal

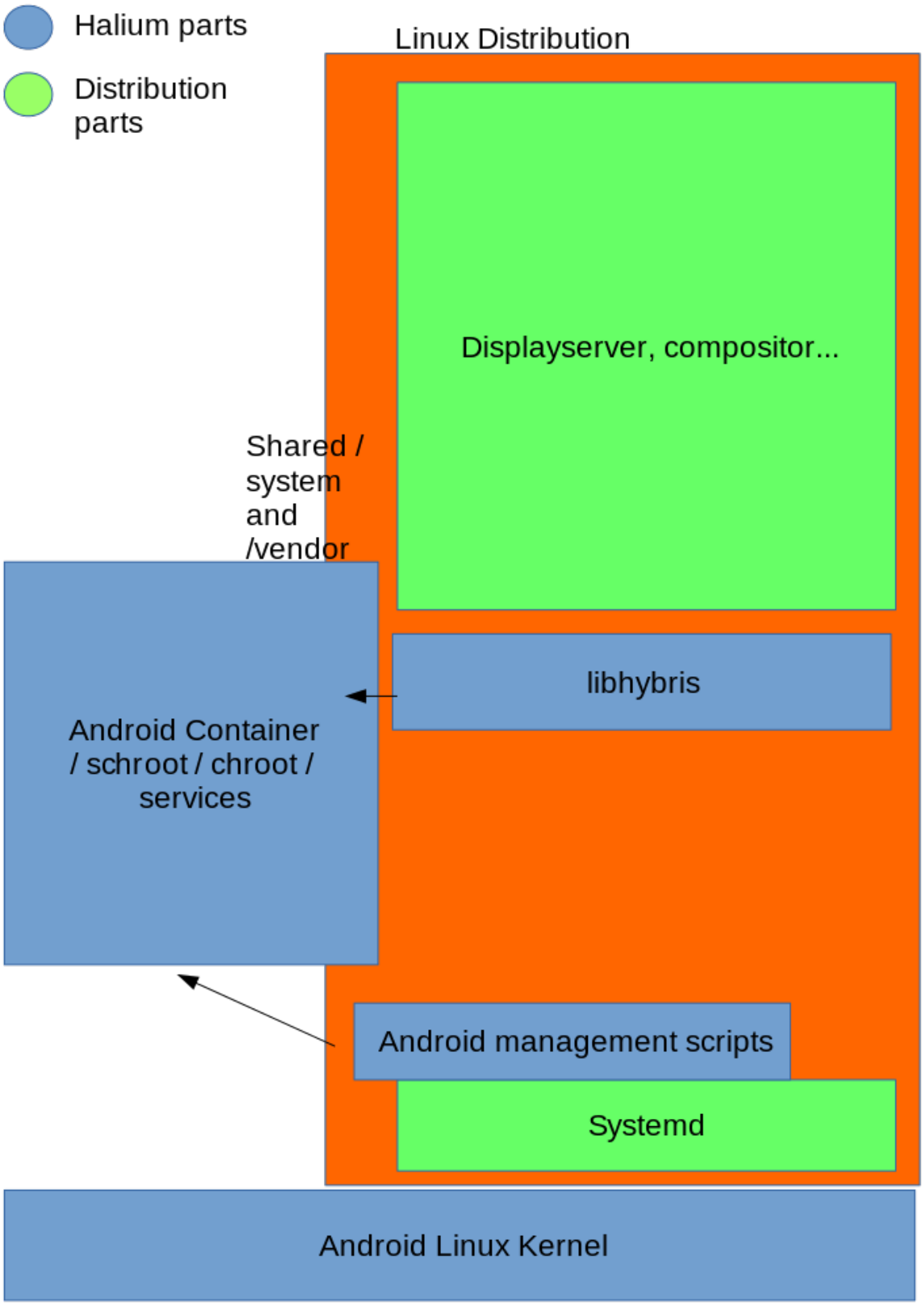
The system is running on a android manufacturer linux kernel, IF the device is mainlined, it can run mainline linux kernel

(Initramfs?)

Systemd start's up the system and the android HAL, which runs either in a schroot, container, or directly on the system.

Graphic output, camera and other sensors are managed by libhybris.

The linux distribution running with this stack can either use packages of libhybris etc. offered by us, if it's compatible, or build it's own


Halium parts
Distribution parts
Linux Distribution
Displayserver, compositor...
Shared / system and /vendor
libhybris
Android Container / schroot / chroot / services
Android management scripts
Systemd
Android Linux Kernel

#### Some random links

- http://www.webos-ports.org/wiki/Halium_Considerations
- http://merproject.org/meetings/mer-meeting/2017/mer-meeting.2017-04-19-08.00.txt
- http://merproject.org/meetings/mer-meeting/2017/mer-meeting.2017-04-19-08.00.log.html

#### Mer-meeting takeaway

Jolla guys potentially interested but too early to commit to this. We need a proper proof-of-concept to show mer could run on top. Sailfish OS community devs of course anyone is welcome.

Jolla was concerned about how our things work with the ODMs way of things. This may not be a problem for community projects like ubports, plasma, et. al. but especially for Jolla since they're dealing straight with the ODMs. Something to take into account when we build our infrastructure. We should make it as flexible as possible so that this would work with them also.

### 1.12.4 Initial halium creator script (locusf rambling)

1. Plug in phone, run adb, which pulls in needed binaries from /system
2. builds halium kernel + the boot selection (still wondering what this could be)
3. boot selector then fetches/runs the actual os inside some runtime (container/switch_root up to debate)

Halium kernel means both the actual kernel + middleware needed in order to have a common libhybris base from the running android system

## 1.13 Development

### 1.13.1 Development plan/outline:

*Goal:* being able to `adb shell` / `telnet` / `SSH` into the `GNU/Linux` system where you can run tests related to `android` hardware enablement and they work out of the box.

*Reference device(s):* `Nexus 5`, `Oneplus one`, `Nexus 5X` (It will be interesting to see how easy it makes it for `Ubuntu Touch` once `Halium` works for this device).

*Reference rootfs:* `Ubuntu ARM` or `ArchLinux ARM` or `Debian ARM` or `Fedora ARM` (Requirement: `systemd` as main init system)

### 1.13.2 Initial development (stage 0, libhybris):

- [ ] Fork the Android source tree with libhybris patches
- [ ] Bring continuous integration system up to build the android image
- [ ] Start with the basic rootfs (ubuntu 16.04)
- [ ] Build and integrate the libhybris (upstream) to the base rootfs

- [ ] Create publishing infrastructure for the distribution of the android image and rootfs
- [ ] At this point it's fine if not all tests "pass" but they should at least run.

### 1.13.3 Hardware enablement (stage 1):

- [ ] After stage 0 we will have a system to work on where we are able to run the `libhybris` tests and get the logs required, in this stage we can pick random hardware component and get it working
- [ ] At end of this stage, Halium is "ready" for distributions

### 1.13.4 Device enablement (stage 2):

- [ ] During this stage we will have most of the infrastructure ready for including new devices, so this is the time to go wild ;-)

*Tests:* We need to write tests that can run in our reference rootfs to make it easier to spot problems, we need both direct tests that live in `android` and tests that use `libhybris`.

*Automated tests:* These tests will run run after each `CI` build to ensure the builds that comes out of our build servers are working. (@UBports has some tests we can use for this)

### 1.13.5 Future ideas

- [ ] Common upstream msm kernel for all devices
- [ ] Common sets of qcom drivers
- [ ] All qcom devices to caf
- [ ] Freedreno for mainlined devices
- [ ] Backport 5.1 and 6.0 kernel driver required by 5.1 and 6.0 blobs
- [ ] Upstream the Android N port of libhybris from UBports fork
- [ ] Common place to put device configs for userspace (not talking about android device configs here, but the config each OS uses to configure different parts)
- [ ] Create a *translator* that converts common configs to OS specific configs
- [ ] Possibly something to "update" kernel from distribution packaging. I remember @mickybart/gnulinux_support has something around that.
- [ ] Create a sandbox env to test *only* the hal and libhybris

## 1.14 Documentation Todo

This is an automatically generated page of all of the "todo" items in this documentation. If you're looking to help us out, feel free to take some on!

---

**Todo:** Mention that the config parameters CONFIG_IKCONFIG and CONFIG_IKCONFIG_PROC need to be set to y, otherwise Halium wont boot (or add them to the check script

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/build-sources.rst, line 66.)

---

**Todo:** Document the debugging of libhybris: https://wiki.ubuntu.com/Touch/Core/UbuntuDebugAndroid

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/debug-build/debug-android-userspace.rst, line 86.)

---

**Todo:** Document how to deal with firmware partitions.

For example xLeEco Le Max2, codename "x2" has a firmware partition where the vendor blobs are stored. Initially lxc@android would not start. The resolution was roughly:

- no need for a vendor blobs repository in the manifest
- determine firmware partition name
- ensure fix-mountpoints takes it into account
- reflash android to ensure the blobs are in the partition
- reflash halium

See http://logs.nslu2-linux.org/livelogs/halium/halium.20180430.txt

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/debug-build/debug-android-userspace.rst, line 89.)

---

**Todo:** Add tests using more heavy graphics applications using Wayland (mir)

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/debug-build/graphics.rst, line 28.)

---

**Todo:** Add information for installing packages on other distros

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/first-steps.rst, line 74.)

---

**Todo:** Distributions like Ubuntu Touch and LuneOS go here. Links to their respective documentation should be sufficient, but please place them inside the toctree above.

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/install-build/index.rst, line 28.)

---

**Todo:** Make sure phablet password is set either by using chroot from recovery or halium-install from JBBgameich.

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/install-build/plasma-mobile.rst, line 22.)

---

**Todo:** Document PM specific debug instructions. Some lose notes copied from irc log:

- run test_XYZ with sudo
- pgrep kwin_wayland
- journalctl | grep simplelogin

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/porting/install-build/plasma-mobile.rst, line 27.)

**Todo:** Add the manifest for Bullhead

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/supplementary/devices/bullhead.rst, line 15.)

**Todo:** Document some other device variants.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/supplementary/devices/cedric.rst, line 9.)

**Todo:** TODO provide a nice link to help users unlock their bootloader

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/supplementary/devices/cedric.rst, line 149.)

**Todo:** Provide some useful resources here.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/supplementary/devices/cedric.rst, line 176.)

**Todo:** Add info about this on Android/Others

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/supplementary/hardware-enablement.rst, line 259.)

**Todo:** create reference-device.rst

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/supplementary/how-to-document.rst, line 53.)

**Todo:** add component_template.rst

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/halium-docs/checkouts/latest/supplementary/how-to-document.rst, line 84.)