HWpack Requirements

Contents

Concepts **2**
- Image Building Stages . . . 2
  - OSpack . . . 2
  - HWpack . . . 3
- Apertis packages . . . 3
- Typical Apertis OS layout . . . 4

HWpack Components **5**
- Bootloader . . . 5
- Linux Kernel . . . 5
- Firmware . . . 6
- Debos .yaml configuration . . . 7
  - Documentation . . . 7

Testing **7**

Licensing **7**

This documentation covers the requirements and considerations that should be taken into account when implementing “hardware packs”for the Apertis project.

Concepts

This section briefly covers the concepts and expectations of the Apertis platform.

Image Building Stages

Apertis images are built from binary packages, packaged in the `.deb` format. Building these packages is expected to be carried out from source by the Apertis infrastructure, ensuring all packages dependencies are properly described and reducing the risk of unexpected dependencies.

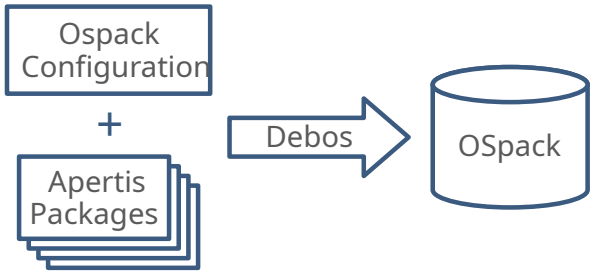
The selection and packaging of these packages are predominantly driven by the needs of the two main process steps required to create images, known as the `OSpack` and `HWpack`.

OSpack

The OSpack stage generates one or more generic (architecture specific but largely hardware independent) archived rootfs built from Apertis packages. These rootfs archives are known as OSpacks. The process is managed by a tool called Debos[1], which uses yaml configuration files to guide what steps it takes. Apertis provides yaml files to assemble a number of differently targeted OSpacks, ranging from a fixedfunction GUI-less OSpack, a HMI focused GUI OSpack and

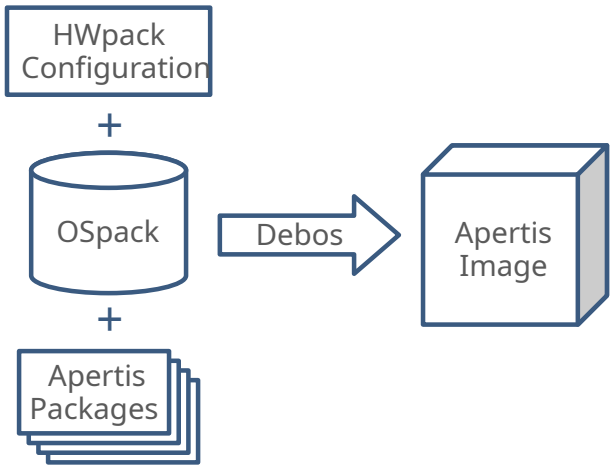
[1] https://github.com/go-debos/debos

a development environment with a desktop style GUI and has pre-packaged the components required to generate these OSpacks.



HWpack

Unlike the OSpack step, the hardware package (HWpack) step does not result in an item known as a HWpack. The HWpack is comprised of a Debos script which controls the processing of a run time determined OSpack to convert it from a hardware independent OSpack into an image which can be successfully booted on a specific hardware platform. In addition to developing the HWpack script, the HWpack step requires the modification and packaging of the required components to perform this transformation.



Apertis packages

Apertis standardizes on a specific set of components (with specific versions) and technologies to fulfill the needs of the target platforms. This maximizes sharing and reuse, thus minimizing effort and cost of maintaining common components across products. Deviations from the standard selection may be needed to accommodate product-specific needs but such deviations tend to reduce reuse and thus increase the long-term maintenance efforts. It will likely fall on the product team in question to carry this added effort. As a result, it is strongly preferred

for deviations to be minimized with generic improvements and additions made to the standard components to add the required functionality where possible.

The components selected as part of the base Apertis system need to meet a number of project criteria.

Licensing requirements Components need to be licensed in such a way that they are acceptable for distribution in target devices. For example, GPL-3 is problematic and thus avoided.

Software revisions The specific revisions of the packages are picked to balance the competing customer needs of having up-to-date versions (and thus features), stability and the need for a strong security road map.

Close to upstream Apertis aims to remain relatively close to its upstreams (where the majority of packages are based on Debian stable, the kernel on the latest LTS release). This minimizes the effort required to migrate to newer versions as it means there are minimal patches to port. A large deviation from upstream also decreases effectiveness of testing and the validity of review performed on upstream versions.

These are some of the key packages from which the Apertis system is built:

- U-Boot/systemd-boot
- Linux kernel
- systemd
- Apparmor
- Wayland
- Mesa
- PulseAudio

All Apertis packages are packaged using standard Debian packaging, with source code and package configuration stored in the Apertis GitLab enabling automation of the package build process.

Typical Apertis OS layout

The reference Apertis images share a common layout per architecture, enabling images to be shared across the various supported platforms of each architecture:

- Bootloader typically stored in flash
- Kernel and other boot components and configuration stored on rootfs (enabling current update mechanism)
- OSTree used as part of update strategy and rollback (non-OSTree options available for development)

It is expected that the requirements and practicalities of products based on Apertis will require deviations to be made from this layout. Such deviations however should be carefully considered. Some, such as storing the bootloader at the beginning of the same medium as the rootfs, carry very little impact as far as the functionality of Apertis is concerned. Others such as using a different

bootloader, storing the kernel outside of the rootfs or using a different update strategy (such as A/B partitioning) may pose in a non-trivial effort for integration, loss of some Apertis functionality and/or extra on-going maintenance effort.

The OSpack is expected to contain common functionality to enable use of supported hardware, for example the OSpacks which are intended to be used with an operational graphical environment include Wayland, though the hardware specific drivers are in the HWpacks. When enabling new types of functionality, it is expected that generic support would be added to the OSpacks where applicable. If such functionality is widely used, this should be integrated into the Apertis OSpacks. Support for niche functionality, or functionality not of general interest to Apertis, will need to be added to a product specific OSpack.

HWpack Components

As with the OSpack (and unless specific exceptions provided) all components should be properly packaged and provided with source to enable debugging, extension and further optimization. It is expected that some changes may be viable to be included in the main Apertis packages, some packages may be added to the main Apertis package repositories and others will need bespoke packages which would typically be stored in a dedicated project area, as described in the contribution process[2] document. It is typical for the following areas to need modifications or to be provided, though other modifications may also be required.

Bootloader

Apertis standardizes on the U-Boot as the bootloader on all non-x86 platforms. In order to support the standard Apertis boot, update and rollback functionality it is necessary for the configuration to include the "Generic Distro Configuration Concept[3]"(often referred to as "Distro Boot"). The configuration used by this mechanism has been tweaked to work with Apertis rollback mechanisms.

In order to enable efficient development, it would be advisable to ensure that access to the boot prompt is enabled along with networking support and the PXE and DHCP boot options where applicable. (Note: U-Boot can supports networking via a USB RNDIS gadget should a USB On-The-Go (USB OTG) port be available.)

Linux Kernel

Apertis expects projects that using it have a need to take product security seriously, as a result known kernel vulnerabilities need to be patched and updates

[2] https://www.apertis.org/policies/contributions/
[3] https://gitlab.apertis.org/pkg/u-boot/blob/apertis/v2020pre/doc/README.distro

made available. Apertis uses and tracks the latest upstream longterm[4] stable (LTS) kernel available at time of a release being made. The Apertis project strongly recommends that when products use their own kernel, these are kept as close to the upstream kernel as possible and preferably based on an LTS kernel.

It is understood that in some circumstances it may be necessary to utilize a heavily modified "vendor kernel". Please note that these kernels are typically not provided with any form of long-term support and thus may quickly lack important security and stability fixes. Unless otherwise agreed, the burden of supporting such kernels will remain with the product team. Likewise, in addition to lacking a source of security fixes, many older kernels are known to have serious vulnerabilities that can only be fully resolved/mitigated by updating to a newer kernel. Apertis strongly discourages the use of such kernels.

The Apertis kernel contains a number of modifications primarily to enhance the Apparmor support provided by the upstream kernel. The patches used by the stock Apertis kernel can be found in the Apertis GitLab[5]. In order to support Apertis'use of Apparmor, a kernel needs to support the following Apparmor mediations:

- file
- ptrace
- signal
- dbus
- network
- capability
- mount
- umount
- namespaces

Additionally, the kernel should be configured to support the functionality required by systemd[6].

For development purposes, the kernel should provide early serial debugging and be capable of booting from an NFS rootfs.

Firmware

It is understood that many hardware platforms may need firmware, provided by the vendor as binaries, to use certain functionality provided by the device. It is still expected that such firmware is packaged as a deb package, though it is understood that source will not be available for such components. The Apertis infrastructure should still be used to build the binary packages.

[4] https://www.kernel.org/category/releases.html
[5] https://gitlab.apertis.org/pkg/linux/tree/apertis/v2020pre/debian/patches/apparmor
[6] https://gitlab.apertis.org/pkg/systemd/blob/apertis/v2020pre/README

Debos .yaml configuration

Apertis uses Debos to automate the conversion of binary packages into images suitable for installation on specific targets in several stages. The configuration used for the Apertis reference platforms can be found in GitLab[7] with their use documented in `README.md`. It is expected that a HWpack provides configuration file(s) that:

- Generate the required image(s) from either a reference or project specific OSpack
- Generate images containing the partitioning expected by the target and project
- Add any extra components needed via the installation of packages
- Are provided with any scripts required to aid in the application of minor changes to tweak the image to required default configuration
- Generate any project specific OSpacks when sufficient support can't be added to the generic OSpack recipes to cover the functionality required by the relevant project.

Documentation

Documentation should be provided with the Debos configuration detailing the use of any configuration files provided and documenting the process to be followed to install the generated images into a new target device to yield a booting system.

Testing

Apertis provides infrastructure to both continuously build and test software on target devices based on Docker[8], GitLab CI/CD[9] and LAVA[10]. It is expected that the provided source and configuration artifacts (and possibly binary firmware as mentioned above), when integrated into the provided Apertis infrastructure, will be capable of generating images which pass hardware boot testing with no manual steps required.

Licensing

Code, including build scripts, helpers and recipes, developed for Apertis should comply with the Apertis Licensing[11].

[7] https://gitlab.apertis.org/infrastructure/apertis-image-recipes
[8] https://www.docker.com/
[9] https://docs.gitlab.com/ee/ci/
[10] https://lavasoftware.org/
[11] https://www.apertis.org/guides/licensing/license-applying/