Sysroots and Devroots

Contents

Sysroots and devroots are two development rootfs meant to provide an environment to build software for Apertis, targeting foreign architecture that don't match the CPU architecture of the build host (for instance, building ARM64 binaries from a Intel-based host).

They are meant to address different use cases with different trade-offs.

Sysroot

Sysroots are file system trees specifically meant for cross-compilation and remote debugging targeting a specific release image.

They are meant to be read-only and target a specific release image, shipping all the development headers and debug symbols for the libraries in the release image.

Sysroots can be used to cross-compile for Apertis from a third-party environment using an appropriate cross-toolchain[1]. They are most suited for early development phases where developers focus on quick iterations and rely on fast incremental builds of their components.

Cross-compilation using sysroot requires support from the project build system, which then needs to be set up to appropriately point to the sysroot and to the cross compiler. Not all build systems support cross compilation and some may require patching to make it work properly.

The Apertis SDK ships the `ade` tool to simplify sysroots management and the configuration of projects based on the GNU Autotools[2] to use them, focusing in particular on application development. See the Apertis Development Environment[3] guide for information on how to use ade.

Sysroots can be used without `ade` by manually downloading the `sysroot` tarball from the release artifact repository and then unpack it locally with `tar`, see the

[1] https://sjoerd.pages.apertis.org/apertis-website/guides/cross-build-toolchain/

[2] https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html

[3] https://sjoerd.pages.apertis.org/apertis-website/guides/ade/

instructions in the cross-toolchain documentation[4] for a full walk-through on using them on non-Apertis hosts.

Since unpacked sysroots are self-contained folders, multiple sysroots can coexist on a single system to target multiple architectures and releases: for instance, a single system could host the `armhf` and `arm64` sysroots for `v2019pre` and the `arm64` one for `v2020dev0` at the same time. Using the portable cross-build toolchain[5] matching the target release is recommended.

Sysroots are available from the Apertis release artifact repository as `sysroot*.tar.gz` tarballs under the `$release/$architecture/sysroot/` folder, for instance `sysroot-apertis-v2019pre-arm64-v2019pre.0.tar.gz`[6] under `v2019pre.0/arm64`[7].

Devroot

Devroots are file system trees meant to offer a foreign architecture build environment via containers and binary emulation via the QEMU user mode.

Using emulation means that, for instance, all the binaries on the ARM64 devroot are ARM64 binaries and QEMU translates them at runtime to execute them on a Intel-based host.

This means that builds under a devroot appear to the build system as native builds and no special support or configuration is needed, unlike for actual cross builds using sysroots.

Devroots ship a minimal set of packages and offer the ability to install all the packages in the Apertis archive using the `apt` tool just like on the Apertis SDK itself.

Due to the nature of foreign architecture emulation they impose a considerable overhead on build times compared to sysroot, but they avoid all the intricacies that cross-building involves and offer the ability to reliably build deb packages targeting foreign architectures.

The Apertis SDK ships the `devroot-enter` tool to set up the container environment needed to work in a unpacked devroot, see the "Programming guidelines" section[8] for information on how to use devroot-enter.

Since devroots are self-contained folders like sysroots, multiple devroots may be installed at the same time on a single host to target multiple releases and architectures.

[4] https://sjoerd.pages.apertis.org/apertis-website/guides/cross-build-toolchain/

[5] https://sjoerd.pages.apertis.org/apertis-website/guides/cross-build-toolchain/

[6] https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/sysroot/sysroot-apertis-v2019pre-arm64-v2019pre.0.tar.gz

[7] https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/sysroot/

[8] https://sjoerd.pages.apertis.org/apertis-website/guides/tooling/#development-containers-using-devroot-enter

Devroots are available from the Apertis release artifact repository as the `ospack*.tar.gz` tarballs under the `$release/$architecture/devroot/` folder, for instance `ospack_v2019pre-arm64-devroot_v2019pre.0.tar.gz`[9] under `v2019pre.0/arm64`[10].

As of `v2019pre`, the Apertis SDK images come with the `armhf` devroot pre-installed.

A comparison

Sysroot

- **Benefits**
  - Fast
  - No special requirements on the system
  - Supports remote debugging by providing symbols matching a specific target images
- **Drawbacks**
  - Only works with build systems explicitly supporting cross-building
  - Cannot be customized

Devroot

- **Benefits**
  - Builds appears as native builds to build systems, avoiding cross-compilation issues
  - Can be fully customized, adding, removing and updating packages
- **Drawbacks**
  - Requires a container to be set up on the host (`systemd-nspawn` is recommended)
  - Binary emulation imposes a significant performance overhead
  - Supporting remote debugging requires additional care to ensure that symbols match the software running on the target image

When to use them

- **For application and agent development building app-bundles: use the sysroot**
  - This is the main use-case for using the sysroot and the `ade` tool is meant to simplify this workflow.
- **For platform development building deb packages: use the devroot**

[9] https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/devroot/ospack_v2019pre-arm64-devroot_v2019pre.0.tar.gz

[10] https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/devroot/

  - Support for cross-building deb packages is spotty, using the devroot with `devroot-enter` provides the most reliable solution in this case and enables developers to install extra dependencies not shipped on Apertis images by default.
- **To cross-build for Apertis from a third-party SDK: use the sysroot**
  - If the build system already supports cross-building, using the sysroot does not pose additional requirements on the third-party SDK, while the devroot requires emulation and a container setup.
- **To build projects not supporting cross-compilation: use the devroot**
  - The devroot is meant to emulate native compilation, side-stepping any cross-compilation issue.
  - On a third-party SDK it is still possible to use the devroot using the `devroot-enter` script[11] as long as the following tools are available and set up:
    * `qemu-arm-static`/`qemu-aarch64-static` (from the `qemu-user-static` package) for foreign binary emulation
    * a `binfmt_misc` setup for transparent usage of `qemu-user-static` (provided by the `binfmt-support` package on Debian-based systems)
    * `systemd-nspawn` (from the `systemd-container` package) for setting up the containerized environment

[11] https://gitlab.apertis.org/apertis/apertis-dev-tools/blob/apertis/v2019pre/tools/devroot-enter