



Sysroots and Devroots

# 1 Contents

2 **Sysroot** **2**

3 **Devroot** **3**

4 **A comparison** **4**

5       Sysroot . . . . . 4

6       Devroot . . . . . 4

7 **When to use them** **4**

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

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

## 13 Sysroot

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

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

19 Sysroots can be used to cross-compile for Apertis from a third-party environ-  
20 nment using an [appropriate cross-toolchain](#)<sup>1</sup>. They are most suited for early  
21 development phases where developers focus on quick iterations and rely on fast  
22 incremental builds of their components.

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

27 The Apertis SDK ships the `ade` tool to simplify sysroots management and the  
28 configuration of projects based on the [GNU Autotools](#)<sup>2</sup> to use them, focusing in  
29 particular on application development. See the [Apertis Development Environ-](#)  
30 [ment](#)<sup>3</sup> guide for information on how to use `ade`.

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

---

<sup>1</sup><https://jwd.pages.apertis.org/apertis-website/guides/cross-build-toolchain/>

<sup>2</sup>[https://www.gnu.org/software/automake/manual/html\\_node/Autotools-Introduction.html](https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html)

<sup>3</sup><https://jwd.pages.apertis.org/apertis-website/guides/ade/>

33 instructions in the [cross-toolchain documentation](#)<sup>4</sup> for a full walk-through on  
34 using them on non-Apertis hosts.

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

40 Sysroots are available from the Apertis release artifact repository as `sys-`  
41 `root*.tar.gz` tarballs under the `$release/$architecture/sysroot/` folder,  
42 for instance [`sysroot-apertis-v2019pre-arm64-v2019pre.0.tar.gz`](#)<sup>6</sup> under  
43 [`v2019pre.0/arm64`](#)<sup>7</sup>.

## 44 Devroot

45 Devroots are file system trees meant to offer a foreign architecture build envi-  
46 ronment via containers and binary emulation via the QEMU user mode.

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

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

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

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

60 The Apertis SDK ships the `devroot-enter` tool to set up the container environ-  
61 ment needed to work in an unpacked devroot, see the [“Programming guidelines”](#)  
62 [section](#)<sup>8</sup> for information on how to use `devroot-enter`.

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

---

<sup>4</sup><https://jwd.pages.apertis.org/apertis-website/guides/cross-build-toolchain/>

<sup>5</sup><https://jwd.pages.apertis.org/apertis-website/guides/cross-build-toolchain/>

<sup>6</sup><https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/sysroot/sysroot-apertis-v2019pre-arm64-v2019pre.0.tar.gz>

<sup>7</sup><https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/sysroot/>

<sup>8</sup><https://jwd.pages.apertis.org/apertis-website/guides/tooling/#development-containers-using-devroot-enter>

66 Devroots are available from the Apertis release artifact repository as  
67 the `ospack*.tar.gz` tarballs under the `$release/$architecture/devroot/`  
68 folder, for instance `ospack_v2019pre-arm64-devroot_v2019pre.0.tar.gz`<sup>9</sup> un-  
69 der `v2019pre.0/arm64`<sup>10</sup>.

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

## 72 A comparison

### 73 Sysroot

- 74 • **Benefits**
  - 75 – Fast
  - 76 – No special requirements on the system
  - 77 – Supports remote debugging by providing symbols matching a specific
  - 78 target images
- 79 • **Drawbacks**
  - 80 – Only works with build systems explicitly supporting cross-building
  - 81 – Cannot be customized

### 82 Devroot

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

## 93 When to use them

- 94 • **For application and agent development building app-bundles:**  
95 **use the sysroot**
  - 96 – This is the main use-case for using the `sysroot` and the `ade` tool is
  - 97 meant to simplify this workflow.
- 98 • **For platform development building deb packages: use the dev-**  
99 **root**

---

<sup>9</sup>[https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/devroot/ospack\\_v2019pre-arm64-devroot\\_v2019pre.0.tar.gz](https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/devroot/ospack_v2019pre-arm64-devroot_v2019pre.0.tar.gz)

<sup>10</sup><https://images.apertis.org/release/v2019pre/v2019pre.0/arm64/devroot/>

- 100           – Support for cross-building deb packages is spotty, using the devroot  
101           with `devroot-enter` provides the most reliable solution in this case  
102           and enables developers to install extra dependencies not shipped on  
103           Apertis images by default.
- 104 • **To cross-build for Apertis from a third-party SDK: use the sysroot**  
105       **root**
    - 106           – If the build system already supports cross-building, using the sysroot  
107           does not pose additional requirements on the third-party SDK, while  
108           the devroot requires emulation and a container setup.
  - 109 • **To build projects not supporting cross-compilation: use the devroot**  
110       **root**
    - 111           – The devroot is meant to emulate native compilation, side-stepping  
112           any cross-compilation issue.
    - 113           – On a third-party SDK it is still possible to use the devroot using the  
114           [devroot-enter script](#)<sup>11</sup> as long as the following tools are available and  
115           set up:
      - 116               \* `qemu-arm-static/qemu-aarch64-static` (from the `qemu-user-static`  
117               package) for foreign binary emulation
      - 118               \* a `binfmt_misc` setup for transparent usage of `qemu-user-static`  
119               (provided by the `binfmt-support` package on Debian-based systems)
      - 120               \* `systemd-nspawn` (from the `systemd-container` package) for setting  
121               up the containerized environment
- 122

---

<sup>11</sup><https://gitlab.apertis.org/apertis/apertis-dev-tools/blob/apertis/v2019pre/tools/devroot-enter>