



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](#)¹. 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](#)² to use them, focusing in
29 particular on application development. See the [Apertis Development Environ-](#)
30 [ment](#)³ 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

¹<https://em.pages.apertis.org/apertis-website/guides/cross-build-toolchain/>
²https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html
³<https://em.pages.apertis.org/apertis-website/guides/ade/>

33 instructions in the [cross-toolchain documentation](#)⁴ 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](#)⁵
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, for
42 instance `sysroot-apertis-v2021-amd64-v2021.0.tar.gz`⁶ under `v2021.0/arm64`⁷.

43 Devroot

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

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

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

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

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

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

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

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

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

⁶<https://images.apertis.org/release/v2021/v2021.0/amd64/sysroot/sysroot-apertis-v2021-amd64-v2021.0.tar.gz>

⁷<https://images.apertis.org/release/v2021/v2021.0/amd64/sysroot/>

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

65 Devroots are available from the Apertis release artifact repository as the
66 `ospack*.tar.gz` tarballs under the `$release/$architecture/devroot/` folder, for
67 instance `ospack_v2021-amd64-devroot_v2021.0.tar.gz`⁹ under `v2021.0/arm64`¹⁰.

68 As of v2019, the Apertis SDK images come with the `armhf` devroot pre-installed.

69 A comparison

70 Sysroot

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

79 Devroot

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

90 When to use them

- 91 • **For application and agent development building app-bundles:**
92 **use the sysroot**
 - 93 – This is the main use-case for using the sysroot and the `ade` tool is
 - 94 meant to simplify this workflow.
- 95 • **For platform development building deb packages: use the dev-**
96 **root**
 - 97 – Support for cross-building deb packages is spotty, using the devroot
 - 98 with `devroot-enter` provides the most reliable solution in this case

⁹https://images.apertis.org/release/v2021/v2021.0/amd64/devroot/ospack_v2021-amd64-devroot_v2021.0.tar.gz

¹⁰<https://images.apertis.org/release/v2021/v2021.0/amd64/devroot/>

99 and enables developers to install extra dependencies not shipped on
100 Apertis images by default.

- 101 • **To cross-build for Apertis from a third-party SDK: use the sys-**
102 **root**
 - 103 – If the build system already supports cross-building, using the sysroot
104 does not pose additional requirements on the third-party SDK, while
105 the devroot requires emulation and a container setup.
- 106 • **To build projects not supporting cross-compilation: use the de-**
107 **vroot**
 - 108 – The devroot is meant to emulate native compilation, side-stepping
109 any cross-compilation issue.
 - 110 – On a third-party SDK it is still possible to use the devroot using the
111 [devroot-enter script](#)¹¹ as long as the following tools are available and
112 set up:
 - 113 * `qemu-arm-static/qemu-aarch64-static` (from the `qemu-user-static`
114 package) for foreign binary emulation
 - 115 * a `binfmt_misc` setup for transparent usage of `qemu-user-static`
116 (provided by the `binfmt-support` package on Debian-based sys-
117 tems)
 - 118 * `systemd-nspawn` (from the `systemd-container` package) for setting
119 up the containerized environment

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