

ZAINIUM DYNAMICS

OPEN SPECIFICATION

Dualayer Hierarchy Standard

A layered filesystem hierarchy in which musl and GNU libc run side by side

VERSION	1.1
PUBLISHED	November 2, 2026
BUILD	2026-11-02 · 55a55d4
LICENSE	CC BY-SA 4.0

ABSTRACT

The Dualayer Hierarchy Standard (DHS) defines where the files of an operating system live when the system is assembled from separate layers: a self-sufficient base, a layer of installed software, and an ignition layer. It also defines how two C libraries — musl and the GNU C Library (glibc) — are installed on the same system, each in a directory of its own, so that programs linked against either one run without conflicting with the other. Either C library can be the one the base is built on; the distribution chooses. The standard is written for operating system vendors, package managers, toolchain authors and application developers.

Copyright © 2026 Dualayer Hierarchy Standard Project Contributors

This standard is licensed under the Creative Commons Attribution-ShareAlike 4.0 International license (CC BY-SA 4.0). You may copy, adapt and redistribute it, including in commercial products, provided that you credit the Dualayer Hierarchy Standard Project and distribute derived versions under the same license. The full license text is in the file `LICENSE` of the source repository and at https://creativecommons.org/licenses/by-sa/4.0/.

All trademarks are the property of their owners. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Table of Contents

1. Introduction . . . 5
1.1. Purpose . . . 5
1.2. The name . . . 5
1.3. Scope . . . 5
1.4. Conventions . . . 6
1.5. Relationship to the FHS . . . 6

2. Design principles . . . 7

3. The root directory . . . 8
3.1. Required entries . . . 8
3.2. Entries that must not exist . . . 8

4. The `/overlayer` hierarchy . . . 9
4.1. Overview . . . 9
4.2. Layer names . . . 9
4.3. `/overlayer/syshub`: the system layer . . . 10
4.4. `/overlayer/userland`: the userland layer . . . 11
4.5. `/overlayer/ignition`: the ignition layer . . . 11
4.6. `/overlayer/agents` . . . 11
4.7. `/overlayer/.syshub-pending` . . . 12
4.8. Documentation at the top of the hierarchy . . . 12

5. Composing the system view . . . 13
5.1. Order at boot . . . 13
5.2. Precedence . . . 13
5.3. Rescue . . . 13
5.4. Live media and installed systems . . . 14

6. Two C libraries side by side: musl and glibc . . . 15
6.1. Why two C libraries . . . 15
6.2. Primary and secondary C library . . . 15
6.3. The two designs . . . 16
6.3.1. Design A: musl primary, glibc secondary . . . 16
6.3.2. Design B: glibc primary, musl secondary . . . 16
6.3.3. Comparison . . . 17
6.4. Directories . . . 17
6.5. Rules . . . 17
6.5.1. The shared directories belong to the primary C library . . . 17
6.5.2. Every program names its own loader . . . 18
6.5.3. Library search paths do not cross . . . 18
6.5.4. Both C libraries read the same system configuration . . . 18

6.5.5. Toolchains . . . 18
6.5.6. Package managers keep each program on its own C library . . . 19
6.6. Conflicts this layout prevents . . . 20

7. Running software built for FHS systems . . . 21
7.1. Path resolution . . . 21
7.2. Loader redirection . . . 21
7.3. Packages from other distributions . . . 22

8. Package management . . . 23
8.1. The base is never overwritten . . . 23
8.2. Receipts . . . 23
8.3. Libraries . . . 23
8.4. Base upgrades . . . 23
8.5. Protected paths . . . 23

A. Reference . . . 25
A.1. FHS directories and their DHS locations . . . 25
A.2. Reference implementation . . . 25
A.3. Glossary . . . 26
A.4. Acknowledgements . . . 26

CHAPTER

1. Introduction

1.1. Purpose

The Filesystem Hierarchy Standard (FHS) describes a single, shared tree: the vendor's programs, locally installed programs and their libraries are placed together under `/usr`, and one C library serves every program on the system. That model has served UNIX-like systems for three decades, but it makes three things hard:

- *Separating what the vendor ships from what the user installs.* When both write into `/usr/bin` and `/usr/lib`, an upgrade of the base system and an install of third-party software can replace each other's files.
- *Upgrading the base as one unit.* A base that is spread across the shared tree is upgraded file by file, while it is running.
- *Having more than one C library.* The paths `/lib`, `/lib64`, `/usr/include` and the dynamic loader location assume a single C library. Installing a second one into the same directories mixes headers, loaders and runtime libraries of two incompatible ABIs.

The Dualayer Hierarchy Standard (DHS) solves these by giving each part of the system its own *layer* — a directory tree with a single owner — and by giving each C library its own directory, named after its target triplet. The running system sees the layers as one tree, composed at boot with OverlayFS.

1.2. The name

Dualayer joins *dual* and *layer*: the system is built from layers, and two C libraries — musl and glibc — live in it side by side. The standard is abbreviated DHS.

1.3. Scope

This standard specifies:

- the entries allowed at the root of the filesystem;
- the layers under `/overlayer`, what each contains and who may change it;
- how the layers are composed into the system view at boot;
- where musl and glibc are installed and how programs linked against each find their loader, libraries, headers and compilers;
- how software written for FHS systems is run;
- the guarantees a package manager has to keep.

It does not specify the contents of configuration files, the init system, the desktop, or the package format.

1.4. Conventions

The key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT** and **MAY** are to be interpreted as described in RFC 2119.

In paths, *`arch`* is the CPU architecture as it appears in a GNU target triplet (for example `x86_64` or `aarch64`) and *`vendor`* is the vendor field of the triplet chosen by the implementation (for example `zainium`). A path written as `/overlayer/syshub/bin` names the directory in the composed system view unless the text says “on disk”.

Notes marked *Implementation note* describe how the reference implementation, Zainium OS, meets a requirement. They are informative, not normative.

1.5. Relationship to the FHS

DHS is not an extension of the FHS: a DHS system has no `/usr`, `/bin`, `/lib` or `/etc` at its root. Programs built for FHS systems still run, because their FHS paths are resolved to DHS paths when they are used (Chapter 7, *Running software built for FHS systems*). Section A.1, “FHS directories and their DHS locations” lists where each FHS directory lives under DHS.

CHAPTER

2. Design principles

The requirements in the following chapters follow from these principles.

One root for the operating system
: Everything the operating system ships or installs lives under `/overlayer`. The root directory holds only that directory, user data and the virtual filesystems the kernel provides.

Every layer has one owner
: The base layer is changed only by base upgrades, the userland layer only by the package manager, the ignition layer only by base upgrades and the installer. No layer writes into another.

The base stands on its own
: The base layer is a complete, bootable system. Deleting the userland layer leaves a working system, never a broken one.

Compose, do not mix
: Layers are combined into one view by stacking them (OverlayFS), not by copying files of one layer into another. Each file on disk belongs to exactly one layer.

Each C library in its own directory
: A C library, its loader, its headers and its toolchain live together in a directory named after the library's target triplet. Two C libraries never share a `lib` or `include` directory.

No compatibility directories at the root
: There is no `/usr`, `/lib64`, `/bin` or `/var`, not even as a symbolic link. Software that names those paths is served by path resolution at run time instead.

Base upgrades are atomic
: A new base is prepared while the system runs and put in place at the next boot, before anything uses it. The running system never sees a half-upgraded base.

CHAPTER

3. The root directory

3.1. Required entries

The root directory of a DHS system **MUST** contain the following entries and **SHOULD NOT** contain others.

ENTRY	CONTENTS
`/overlayer`	The operating system: all layers (Chapter 4, *The `/overlayer` hierarchy*).
`/home`	User home directories.
`/root`	The home directory of the root user.
`/dev`	Device files (devtmpfs).
`/proc`	Process information (procfs).
`/sys`	Kernel and device information (sysfs).
`/run`	Run-time state since boot (tmpfs).
`/tmp`	Temporary files.

3.2. Entries that must not exist

The following FHS directories **MUST NOT** exist at the root, neither as directories nor as symbolic links: `/usr`, `/bin`, `/sbin`, `/lib`, `/lib32`, `/lib64`, `/libx32`, `/etc`, `/var` and `/boot`. Their contents live in the layers; Section A.1, “FHS directories and their DHS locations” lists where.

Keeping these names absent is what makes the layers enforceable: a program cannot write into `/usr/lib` and thereby into the base, because there is no such directory to write into.

CHAPTER

4. The `/overlayer` hierarchy

4.1. Overview

On disk, `/overlayer` holds the layers side by side. The tree below shows a system whose primary C library is musl; Section 6.3, "The two designs" shows the same tree with glibc as the primary C library.

```
/overlayer/
├── syshub/                       system layer: the base operating system
│   ├── bin/  sbin/  libexec/     programs
│   ├── lib/                      shared libraries of the primary C library
│   ├── include/                  headers for the libraries in lib/
│   ├── share/                    architecture-independent data
│   ├── etc/                      system configuration
│   ├── var/                      variable state of base services
│   ├── arch-vendor-linux-musl/   primary C library: loader, libc, headers, toolchain
│   └── drivers/
│       ├── modules/kernel/       kernel modules
│       └── hardware/firmwares/   device firmware
├── userland/                     userland layer: software the user installs
│   ├── union/                    files of installed packages
│   │   ├── var/                  variable data of installed software
│   │   └── arch-vendor-linux-gnu/   secondary C library, when installed here
│   ├── work/                     OverlayFS work directory
│   └── registry/                 package receipts, index, caches, keys
├── ignition/                     ignition layer (read-only)
│   ├── kernel/                   kernels and initramfs images
│   └── bootloader/             boot loader files
├── agents/                       instructions for AI agents (optional)
└── .syshub-pending/              a staged base upgrade (only while one waits)
```

In the running system `/overlayer/syshub` is the composed view of the system and userland layers (Chapter 5, *Composing the system view*); the other entries are visible at their own paths.

4.2. Layer names

The names `syshub`, `userland` and `ignition` are the standard names of the three layers, and `.syshub-pending` the standard name of the staging directory. An implementation **SHOULD** use them. An implementation **MAY** give the layers names of its own; it then **MUST** keep the structure defined here unchanged inside each layer, and **MUST** document its names in `/overlayer` (Section 4.8, "Documentation at the top of the hierarchy"). Everything else in this standard — the root directory, the directories inside each layer, the C library directories and the rules — is the same whatever the layers are called.

LAYER	DHS NAME	ZAINIUM OS NAME
System (base)	`/overlayer/syshub`	`/overlayer/syshub` (same)
Userland	`/overlayer/userland`	`/overlayer/zexlib` (Zainium Executable Library)
Ignition (boot)	`/overlayer/ignition`	`/overlayer/zaisys`
Staged base upgrade	`/overlayer/.syshub-pending`	`/overlayer/.syshub-pending` (same)
Agent instructions	`/overlayer/agents`	`/overlayer/agents` (same)

4.3. `/overlayer/syshub`: the system layer

The system layer is the base operating system as the distribution ships it: C library, toolchain, init system, services, desktop and drivers. It **MUST** boot and run with the userland layer absent. It **MUST** be changed only by a base upgrade (Section 8.4, “Base upgrades”).

`bin, sbin`
: Programs. `sbin` holds programs meant for the administrator. Both are on the default `PATH`.

`libexec`
: Helper programs run by other programs, not by users.

`lib`
: Shared and static libraries built for the primary C library, and the data directories that FHS places under `/usr/lib` (for example `lib/systemd`, `lib/udev`, `lib/pkgconfig`). A library built for any other C library **MUST NOT** be placed here.

`include`
: Headers for the libraries in `lib`. The C library's own headers are in its triplet directory, not here.

`share`
: Architecture-independent data: icons, locales, manual pages, desktop files, schemas.

`etc`
: System configuration, including the user and group databases, name service, host names and the time zone link. Every C library on the system **MUST** read these files from here (Section 6.5.4, “Both C libraries read the same system configuration”).

`var`
: Variable data of base services: logs, caches, spool, the state of the service manager.

`arch``-`*`vendor`*`-linux-`*`libc`*
: The primary C library with its loader, headers and cross-named toolchain, and the secondary C library if the distribution ships it in the base (Chapter 6, *Two C libraries side by side: musl and glibc*).

`drivers/modules/`*`kernel`*
: Kernel modules of each installed kernel (FHS: `/lib/modules`).

drivers/hardware/firmwares
: Firmware loaded by drivers (FHS: `/lib/firmware`).

The list of base packages and the package mirrors are kept in a directory of the system layer chosen by the implementation.

> **IMPLEMENTATION NOTE**
>
> Zainium OS keeps them in `/overlayer/syshub/engine/services` (`syshub.toml`, `zex_services.toml`).

4.4. `/overlayer/userland`: the userland layer

The userland layer holds everything the user installs on top of the base. It is changed only by the package manager.

union
: The files of installed packages, in the same layout as the system layer (`bin`, `lib`, `etc`, `share`, …). This directory is the upper layer of the system view, so its files appear in `/overlayer/syshub`. A secondary C library installed as a package lives in its triplet directory here (Chapter 6, *Two C libraries side by side: musl and glibc*).

  `union/var` holds the variable data of installed software, for example a system-wide Flatpak installation in `union/var/lib/flatpak`. Software whose storage cannot live on OverlayFS, such as an OSTree repository, **MUST** use it by its path under the userland layer, not through the system view.

work
: The OverlayFS work directory belonging to `union`. It **MUST** be on the same filesystem as `union` and **MUST NOT** be used for anything else.

registry
: One receipt per installed package listing every file it installed, the package index, download caches and trusted signing keys.

4.5. `/overlayer/ignition`: the ignition layer

The ignition layer holds what is needed before the root is mounted: kernels, initramfs images and the boot loader (FHS: `/boot`). It is mounted read-only in the running system. On firmware that can read only FAT filesystems, the installer **MAY** copy the kernel and initramfs to the EFI system partition; the copies in the ignition layer remain the reference.

4.6. `/overlayer/agents`

An optional directory with instructions, skills and policies for AI coding agents working on the system. Agents **SHOULD** read it before changing the system.

4.7. `/overlayer/.syshub-pending`

A base upgrade that has been downloaded and verified but not yet applied. It exists only while an upgrade waits for the next boot, is writable only by the package manager running as root, and **MUST** be applied or discarded before the system view is composed.

4.8. Documentation at the top of the hierarchy

An implementation **SHOULD** place a short description of the system, its architecture, its layer names and its package manager directly in `/overlayer`, so that the layout explains itself to users and agents.

> IMPLEMENTATION NOTE
>
> Zainium OS ships `/overlayer/README.zainium`, `/overlayer/architecture.md` and `/overlayer/zex.md`.

CHAPTER

5. Composing the system view

5.1. Order at boot

The initramfs **MUST** prepare the system in this order:

1. Mount the root filesystem (a disk partition, or a read-only image on live media).
2. If `/overlayer/.syshub-pending` holds an upgrade, apply it to `/overlayer/syshub` on disk. If a rollback was requested, undo the last applied upgrade instead. Nothing may use the base before this step is complete.
3. Mount the system view at `/overlayer/syshub` with OverlayFS, using:
   - lower layers, highest first: `overlayer/syshub` (the base) and `overlayer/ignition`;
   - upper layer: `overlayer/userland/union`;
   - work directory: `overlayer/userland/work`.
4. Mount `ignition` (read-only), `userland`, `agents` and the other entries of `/overlayer` at their own paths, and `/home` and `/root` from the disk.
5. Switch to the new root and start the init system.

5.2. Precedence

Because the userland is the upper layer, a file it holds at the same path as a base file would hide the base file. The package manager therefore **MUST NOT** install a file at a path the base already provides (Section 8.1, “The base is never overwritten”). Configuration that an administrator edits in `/overlayer/syshub/etc` is written to the upper layer by OverlayFS and survives base upgrades.

5.3. Rescue

An implementation **SHOULD** provide a kernel command line option that boots the base without the userland layer. Because the base stands on its own, this always yields a working system.

> IMPLEMENTATION NOTE
>
> Zainium OS uses `zainium.overlay=off`, and `rd.break=pre-mount|pre-overlay|pre-pivot` to stop in a shell at a given step.

5.4. Live media and installed systems

	LIVE MEDIA	INSTALLED SYSTEM
Root	read-only compressed image in `ignition`	disk partition
Userland layer	in memory, lost at reboot	on disk
`/home`, `/root`	in memory over the image	on disk
Base upgrades	not available	staged, applied at the next boot

CHAPTER

6. Two C libraries side by side: musl and glibc

6.1. Why two C libraries

Every dynamically linked program on Linux is bound to one C library: its ELF header names that library's dynamic loader, and its code calls that library's ABI. The two C libraries in wide use are not interchangeable:

`musl`
: Small, strict and simple to audit; static linking works well; a clean base to build a whole operating system from source. Software built for Alpine, Void (musl) or Chimera links against it.

`GNU C Library (glibc)`
: The C library of most Linux distributions. Prebuilt software — commercial applications, games, vendor SDKs and drivers, many language runtimes and their binary packages — is published for glibc and does not run on musl.

On an FHS system both would compete for `/lib`, `/lib64`, `/usr/include`, the loader path, **`ldd`** and the compiler. DHS gives each its own directory and rules for which programs use which, so a distribution can be built on either C library and still run software built for the other, without either breaking the other.

6.2. Primary and secondary C library

- The *primary* C library is the one the system layer is built against. Each distribution chooses it: musl or glibc. It always lives in the system layer.
- The *secondary* C library is the other one. It is optional and lets the system run software built for it.

> **WHERE THE SECONDARY C LIBRARY LIVES VARIES BY DISTRIBUTION**
>
> A distribution decides whether it ships the secondary C library in the system layer (installed on every system, so base programs may use it) or offers it as a userland package (installed only when the user needs it). Both are conforming. In either case the secondary C library lives in its own triplet directory and follows the rules of Section 6.5, “Rules”; only the layer differs. If the secondary C library is a userland package, the system layer **MUST NOT** contain files linked against it, and a base upgrade **MUST** reject such files, so that removing the package cannot break the base.

6.3. The two designs

6.3.1. Design A: musl primary, glibc secondary

The base is built against musl. glibc is added for prebuilt glibc software.

```
/overlayer/
├── syshub/
│   ├── bin/  lib/  include/  …        musl programs and libraries
│   ├── arch-vendor-linux-musl/   primary
│   │   ├── lib/ld-musl-arch.so.1  lib/libc.so
│   │   ├── include/
│   │   └── bin/                        musl toolchain (cross-named)
│   └── arch-vendor-linux-gnu/    secondary, when shipped in the base
└── userland/union/
    └── arch-vendor-linux-gnu/    secondary, when offered as a package
        ├── lib/ld-linux-arch.so.2  lib/libc.so.6  lib/libstdc++.so.6 …
        ├── include/
        ├── bin/                        ldd, iconv, locale, gnu toolchain
        └── etc/                        ld.so.conf, ld.so.cache
```

IMPLEMENTATION NOTE

Zainium OS uses design A with glibc as a userland package (**`glibc`**, **`glibc-dev`**, **`glibc-toolchain`**) in `/overlayer/zexlib/union/x86_64-zainium-linux-gnu`; the base rejects files linked against glibc.

6.3.2. Design B: glibc primary, musl secondary

The base is built against glibc, as most distributions are today. musl is added for software built for musl distributions, or for building small static programs.

```
/overlayer/
├── syshub/
│   ├── bin/  lib/  include/  …        glibc programs and libraries
│   ├── arch-vendor-linux-gnu/    primary
│   │   ├── lib/ld-linux-arch.so.2  lib/libc.so.6 …
│   │   ├── include/
│   │   ├── bin/                        ldd, iconv, locale, gnu toolchain
│   │   └── etc/                        ld.so.conf, ld.so.cache
│   └── arch-vendor-linux-musl/   secondary, when shipped in the base
└── userland/union/
    └── arch-vendor-linux-musl/   secondary, when offered as a package
        ├── lib/ld-musl-arch.so.1  lib/libc.so
        ├── include/
        ├── bin/                        musl toolchain
        └── etc/                        ld-musl-arch.path
```

6.3.3. Comparison

	DESIGN A (MUSL PRIMARY)	DESIGN B (GLIBC PRIMARY)
Base built against	musl	glibc
Programs in the shared `bin`, libraries in the shared `lib`	musl	glibc
`cc`, **`gcc`**, **`ldd`** on `PATH`	musl's	glibc's
Secondary C library	glibc, in the system layer or as a userland package	musl, in the system layer or as a userland package
Secondary compiler	*`arch`*`-`*`vendor`**`-linux-gnu-gcc`**	*`arch`*`-`*`vendor`**`-linux-musl-gcc`**
Runs without the secondary installed	software built for musl distributions	software built for glibc distributions

6.4. Directories

Each C library is installed into a directory named after its GNU target triplet: *`arch`*`-`*`vendor`**`-linux-musl`** for musl, *`arch`*`-`*`vendor`**`-linux-gnu`** for glibc. The primary one is in the system layer; the secondary one is in the system layer or in `/overlayer/userland/union`.

INSIDE THE TRIPLET DIRECTORY	MUSL	GLIBC
Loader	`lib/ld-musl-`*`arch`*`.so.1`	`lib/ld-linux-x86-64.so.2` (name per architecture)
C library	`lib/libc.so`	`lib/libc.so.6` and the other glibc libraries
Headers	`include`	`include`
Utilities	`bin`	`bin` (**`ldd`**, **`iconv`**, **`locale`**, …)
Loader configuration	`etc/ld-musl-`*`arch`*`.path` (or built in)	`etc/ld.so.conf`, `etc/ld.so.cache`, `etc/ld.so.preload`
C++ runtime	`libstdc++` and `libgcc_s`: the primary's copies in the shared `lib`, the secondary's in its own `lib`	

6.5. Rules

6.5.1. The shared directories belong to the primary C library

The secondary C library **MUST NOT** install anything into the shared `bin`, `sbin`, `lib` or `include` of the system view. As a result **`cc`**, the headers in the default search path, **`ldd`** and every library in `/overlayer/syshub/lib` belong to the primary C library, and installing the secondary changes none of them.

6.5.2. Every program names its own loader

The ELF interpreter (`PT_INTERP`) of every dynamically linked program **MUST** be the DHS path of the loader of the C library it was linked against. A program therefore always starts with the right loader, whichever C libraries are installed.

6.5.3. Library search paths do not cross

- The primary loader searches, in this order, `/overlayer/syshub/lib`, the primary triplet `lib` directory and `/overlayer/userland/union/lib`. Base programs **SHOULD** also carry `/overlayer/syshub/lib` as their library path (`RPATH`).
- The secondary loader **MUST** search only its own `lib` directory and the directories named in its own configuration. It **MUST NOT** search `/overlayer/syshub/lib`: the libraries there are built for the primary C library and cannot be loaded into a process of the other.
- Libraries that programs of the secondary C library need — including its C++ runtime — are installed into the secondary's `lib` directory.

6.5.4. Both C libraries read the same system configuration

Users, groups, passwords, host names, DNS and the time zone are system facts, not library facts. The paths compiled into every C library on the system **MUST** point at the system layer:

COMPILED-IN PATH	DHS LOCATION
`/etc/passwd`, `group`, `shadow`, `hosts`, `resolv.conf`, `nsswitch.conf`, `localtime`	`/overlayer/syshub/etc/…`
Time zone database	`/overlayer/syshub/share/zoneinfo`
`/bin/sh` (used by `system`, `popen`)	`/overlayer/syshub/bin/sh`
Default `PATH` (used by `execvp`)	`/overlayer/syshub/bin:/overlayer/userland/union/bin`
`/etc/mtab`	`/proc/self/mounts`
Loader configuration (`ld.so.conf`, `ld.so.cache`, `ld-musl-`*`arch`*`.path`)	the C library's own `etc`

A C library **SHOULD** be built with these paths changed by source patches, so that no symbolic links are needed.

6.5.5. Toolchains

- The unprefixed compiler names (**`cc`**, **`gcc`**, **`g++`**, **`c99`**) **MUST** belong to the primary C library.
- The secondary C library's compiler **MUST** be installed under its triplet name only, **MUST** take its headers, start files and libraries from the secondary's directory, and **MUST** write the secondary loader's DHS path into the programs it links.

- Each toolchain has its own **`pkg-config`** search path, so that a build against one C library never picks up `.pc` files of the other.
- An implementation **MAY** let the user select which toolchain the unprefixed names run, per shell or as a system setting. The default **MUST** be the primary C library.

> IMPLEMENTATION NOTE
>
> On Zainium OS, **`sys-resolver env glibc`** prints the environment that makes **`cc`**, **`gcc`**, **`g++`**, **`cpp`** and **`pkg-config`** use glibc in the current shell; **`elevate sys-resolver set glibc|musl`** changes the system default and **`sys-resolver get`** shows it. The variable `ZAINIUM_LIBC` overrides the setting for one command.

6.5.6. Package managers keep each program on its own C library

A package manager that adapts foreign packages to DHS paths (Section 7.3, “Packages from other distributions”) **MUST** recognise which C library each program and library was linked against — by its interpreter or its dependency on `libc.so.6` or `libc.so` — and **MUST NOT** rewrite its loader or library path to the other C library's.

6.6. Conflicts this layout prevents

CONFLICT	WHAT GOES WRONG WHEN BOTH SHARE FHS PATHS	HOW DHS PREVENTS IT
Loader	Both loaders and both `libc` files compete for `/lib` and `/lib64`; a package of one library can overwrite the other's.	Each loader is in its own triplet directory; every program names its own (Section 6.5.2, “Every program names its own loader”).
Headers	Compiling against one library's `stdio.h` and linking against the other's `libc` produces programs that fail at run time.	Each library's headers are in its own `include`; each compiler uses only its own.
Shared libraries	A program of one library finds a library built for the other in `/usr/lib` and crashes or fails to load.	The secondary loader never searches the shared `lib` (Section 6.5.3, “Library search paths do not cross”).
C++ runtime	`libstdc++` and `libgcc_s` built for one library are loaded by programs of the other.	Each library has its own copies.
Loader cache	**`ldconfig`** lists libraries of both in one `ld.so.cache`.	Each loader's configuration lives in its own directory and lists only its own libraries.
Tools on `PATH`	The secondary's **`ldd`** or **`iconv`** replaces the primary's and gives wrong answers for base programs.	The secondary's tools stay in its own `bin`, off `PATH`.
`pkg-config`	A build picks up `.pc` files of the wrong library.	Separate search paths per toolchain.
System configuration	Two libraries read users, DNS or the time zone from different places and disagree.	Both read the system layer's `etc` (Section 6.5.4, “Both C libraries read the same system configuration”).
Prebuilt binaries	A program built on another distribution names `/lib64/ld-linux-x86-64.so.2` or `/lib/ld-musl-x86_64.so.1`, which a DHS root does not have.	The missing loader path is redirected to the installed loader of the same C library (Section 7.2, “Loader redirection”).

CHAPTER

7. Running software built for FHS systems

Most software assumes FHS paths: scripts start with `#!/usr/bin/env`, programs open `/etc/passwd` or `/usr/share/…`, and binaries built elsewhere name `/lib64/ld-linux-x86-64.so.2` as their loader. A DHS system runs such software in two ways: packages are adapted when they are installed, and paths that are still FHS-shaped are resolved when they are used.

7.1. Path resolution

An implementation **SHOULD** provide a path resolver that takes effect when a program opens, executes or inspects an FHS path that does not exist. The resolver applies only to paths whose first component is one of `bin`, `sbin`, `lib`, `usr`, `opt`, `var`, `etc` or `boot`; `/home`, `/tmp` and the virtual filesystems are never changed. It rewrites the path as follows and uses the first candidate that exists:

1. Remove a leading `/usr/local/` or `/usr/`.
2. Replace `lib64` with `lib`.
3. Map `lib/modules` to `drivers/modules`, `lib/firmware` to `drivers/hardware/firmwares` and `boot` to `kernel`.
4. Try the result under, in order, `/overlayer/syshub`, the primary C library's directory, `/overlayer/userland/union` and `/overlayer/ignition`.

For example, `/usr/bin/env` becomes `/overlayer/syshub/bin/env`, `/etc/passwd` becomes `/overlayer/syshub/etc/passwd` and `/usr/include/zlib.h` becomes `/overlayer/syshub/include/zlib.h`.

The resolver **MUST** work in programs of both C libraries and therefore **SHOULD NOT** depend on either C library itself.

> IMPLEMENTATION NOTE
>
> Zainium OS implements the resolver as `libsys_resolver.so`, preloaded into service manager units, login shells and PAM sessions. A preloaded library cannot act in statically linked or setuid programs, because the loader does not preload into those.

7.2. Loader redirection

When a program is executed whose interpreter path does not exist — for example `/lib64/ld-linux-x86-64.so.2` or `/lib/ld-musl-x86_64.so.1` — the implementation **SHOULD** start it with the installed loader of the same C library, passing the program as an argument and keeping its name as `argv[0]`. If the matching C library is not installed, the program fails as it would on any system without that library.

7.3. Packages from other distributions

A package manager that installs packages built for other distributions **SHOULD** adapt them when installing: set the loader and library path of each program to the DHS paths of the C library it was linked against, rewrite `pkg-config` prefixes and script interpreters (`#!` lines) to DHS paths, and place the files under `/overlayer/userland/union`, never moving a program to the other C library (Section 6.5.6, “Package managers keep each program on its own C library”).

CHAPTER

8. Package management

DHS does not prescribe a package format. Any package manager that installs into a DHS system **MUST** keep the guarantees of this chapter.

8.1. The base is never overwritten

A userland package **MUST NOT** place a file at a path the base already provides. Such files are left out of the installation. This keeps the base intact under the overlay and makes the userland removable.

8.2. Receipts

For every installed package the package manager **MUST** record a receipt in `/overlayer/userland/registry` that lists each file it installed. Removing the package **MUST** delete exactly those files.

8.3. Libraries

Before installing a package, the package manager **SHOULD** check that every shared library its programs need is provided by the system or by a package being installed with it, and ask the user before installing otherwise.

8.4. Base upgrades

1. New base packages are downloaded and verified while the system runs, and staged in `/overlayer/.syshub-pending`. The staging directory is writable only by the package manager running as root and is protected against change while it waits.
2. At the next boot the initramfs applies the staged upgrade to `/overlayer/syshub` before the system view is composed (Section 5.1, “Order at boot”).
3. The previous state **SHOULD** be kept so that the last upgrade can be rolled back from the boot menu or the kernel command line.

8.5. Protected paths

The base and ignition layers **SHOULD** be protected against accidental change even by the administrator: the privilege tool **SHOULD** refuse destructive commands (removing, moving, overwriting, shredding) on `/overlayer/syshub`, `/overlayer/ignition`, `/overlayer/.syshub-pending` and `/overlayer` itself. The base changes only through base upgrades.

> **IMPLEMENTATION NOTE**
>
> Zainium OS replaces **sudo** with **elevate**, which refuses **rm**, **rmdir**, **mv**, **cp**, **rsync**, **shred** and **unlink** on those paths.

APPENDIX

A. Reference

A.1. FHS directories and their DHS locations

FHS	DHS
/bin, /usr/bin	/overlayer/syshub/bin
/sbin, /usr/sbin	/overlayer/syshub/sbin
/lib, /lib64, /usr/lib, /usr/lib64	/overlayer/syshub/lib (libraries of the primary C library)
/usr/libexec	/overlayer/syshub/libexec
/usr/include	/overlayer/syshub/include
/usr/share	/overlayer/syshub/share
/etc	/overlayer/syshub/etc
/var	/overlayer/syshub/var
/boot	/overlayer/ignition/kernel and the boot loader directory
/lib/modules	/overlayer/syshub/drivers/modules
/lib/firmware	/overlayer/syshub/drivers/hardware/firmwares
/usr/local, /opt	/overlayer/userland/union (installed software, through the package manager)
Loader in /lib or /lib64	the C library's triplet directory (Section 6.4, “Directories”)
/home, /root, /tmp, /run, /dev, /proc, /sys	unchanged

`/srv`, `/mnt` and `/media` are not defined by this version of the standard.

A.2. Reference implementation

Zainium OS is the reference implementation of DHS. Its components map onto the standard as follows:

REQUIREMENT	ZAINIUM OS COMPONENT
Layer names (Section 4.2, “Layer names”)	`syshub`, `zexlib` (userland), `zaisys` (ignition)
C library design (Section 6.3, “The two designs”)	Design A: musl primary, glibc as a userland package
Composition at boot, staged upgrades, rollback	**`quantra-ramfs`** (initramfs)
Package management, receipts, base protection	**`zex`**
Path resolution, loader redirection, toolchain selection	`libsys_resolver.so`, **`sys-resolver`**
Protected paths	**`elevate`**, **`elevate-pam`**
Primary C library and toolchain	musl, **`gcc-musl`**, **`binutils`**
Secondary C library and toolchain	**`glibc`**, **`glibc-dev`**, **`glibc-toolchain`**

Sources: https://github.com/Zainium-Dynamics.

A.3. Glossary

`Layer`
: A directory tree under `/overlayer` with a single owner.

`Base`
: The layer `/overlayer/syshub` on disk: the complete operating system as shipped.

`Userland`
: The layer `/overlayer/userland`: software the user installs.

`System view`
: The OverlayFS mount at `/overlayer/syshub` that combines the base and the userland.

`Target triplet`
: The GNU name of a platform, *`arch`*`-`*`vendor`*`-os-`*`libc`*, used to name a C library's directory.

`Primary / secondary C library`
: The C library the system layer is built against, musl or glibc as the distribution chooses, and the optional other one.

A.4. Acknowledgements

DHS builds on three decades of work on the Filesystem Hierarchy Standard, now maintained by freedesktop.org (https://specifications.freedesktop.org/fhs/). The build system of this document is derived from the FHS project's; see the file `Attributions` in the source repository.