Recent Android Kernel trend for Device Longevity

2024.09.05 Youngmin Nam
(ym0914@gmail.com)
(youngmin.nam@samsung.com)

EU ecodesign regulation

The EU Ecodesign Regulation (Regulation (EU) 2023/1670) includes specific requirements related to smartphone software that are designed to enhance the longevity and sustainability of these devices. Here are the main aspects:

1. **Software Update Requirement**: Manufacturers are obligated to provide software updates, including security patches, for a minimum of five years after a smartphone model is released on the market. This requirement ensures that smartphones remain secure and functional over an extended period, reducing the necessity for early replacement due to outdated software.
2. **Performance After Updates**: The regulation also emphasizes that software updates should not degrade the performance or functionality of the smartphone. This is to prevent scenarios where updates might lead to slower performance, which could push users to replace their devices prematurely.
3. **Transparency and Availability**: Manufacturers must inform consumers about the software support duration and the availability of updates at the time of purchase. This transparency helps consumers make informed decisions based on the expected software lifespan of the device.

These provisions are part of the broader goals of the Ecodesign Regulation to reduce electronic waste, promote sustainable consumption, and extend the usable life of smartphones (EUR-Lex) (Energy Efficient Products) (EUR-Lex).

What is Device Longevity ?

Device longevity refers to the length of time an electronic device or gadget remains functional and performs well. It encompasses various aspects such as durability, reliability, and the overall lifespan of the device. Factors influencing device longevity include manufacturing quality, usage patterns, environmental conditions, and maintenance.

Key elements affecting device longevity include:

1. **Manufacturing Quality:** The materials used and the construction of the device play a significant role. High-quality components and robust assembly can extend a device's life.
2. **Usage Environment:** Conditions such as temperature, humidity, dust, and exposure to physical shocks can impact how long a device lasts.
3. **Frequency of Use:** Devices used more frequently or intensively tend to wear out faster.
4. **Maintenance:** Regular care and maintenance, such as cleaning and timely repairs, can help prolong a device's life.
5. **Software Support:** For smart devices, ongoing software updates can enhance performance, address security vulnerabilities, and extend usability.

Pixel 1 (2016)

Model	Announced	Release		Discontinued	Support		Lifespan[a]
		Date	With OS		Date[1]	Final OS[2]	
Pixel / XL	October 4, 2016	October 20, 2016	7.1 Nougat	April 11, 2018	December 2, 2019	10	3 years, 1 month

android / kernel / msm / **refs/heads/android-msm-marlin-3.18-pie-qpr2**

`665c9a1` Merge branch 'android-msm-marlin-3.18-pi-qpr1' into android-msm-marlin-3.18-pi-qpr2 by Miguel de Dios 6 years ago android-msm-marlin-3.18-pie-qpr2 android-9.0.0_r0.64 android-9.0.0_r0.71

`c31e2d7` Merge branch 'android-msm-marlin-3.18-pi' into android-msm-marlin-3.18-pi-qpr1 by Miguel de Dios · 6 years ago

`eccd578` Merge branch 'android-msm-marlin-3.18-pi-security-next' into android-msm-marlin-3.18-pi by Miguel de Dios · 6 years ago

`d3d9fdf` Revert "Revert "msm: vidc: ignore processing responses in invalid state"" by Miguel de Dios · 6 years ago

`e06da15` Revert "msm: vidc: ignore processing responses in invalid state" by Petri Gynther · 6 years ago

`5a19ffd` UPSTREAM: binder: fix race that allows malicious free of live buffer by Todd Kjos · 6 years ago

`42abdbd` binder: create node flag to request sender's security context by Todd Kjos · 6 years ago

`04a1807` msm: vidc: do not set video state to DEINIT very early by c_darssr · 6 years ago

`aeac614` qcacld-2.0: Integer overflow in wma_unified_link_peer_stats_event_handler by jitiphil · 6 years ago

`d7af6a1` qcacld-2.0: Fix OOB write in wma_extscan_change_results_event_handler by Sunil Ravi · 6 years ago

iPhone 6S (2015)

Model	Release(d)		Discontinued	Support			
	With OS	Date		Ended	Final OS[a]	Lifespan[b] Max[c]	Lifespan[b] Min[d]
iPhone 6s / 6s Plus	iOS 9.0	September 25, 2015	September 12, 2018	August 17, 2022 (last security update: July 29, 2024)	iOS 15.6.1 (15.8.3)	8 years, 10 months	**5 years, 10 months**
iPhone SE (1st)	iOS 9.3	March 31, 2016	September 12, 2018			8 years, 3 months	**5 years, 10 months**

9.2 beta	Darwin Kernel Version 15.0.0: Sun Oct 18 23:34:30 PDT 2015; root:xnu-3248.20.33.0.1~7\/RELEASE_ARM64_S8000
10.2 beta 3	Darwin Kernel Version 16.3.0: Mon Nov 7 22:58:42 PST 2016; root:xnu-3789.30.92~36\/RELEASE_ARM64_S8000
11.2 beta 2	Darwin Kernel Version 17.3.0: Sun Oct 29 17:18:38 PDT 2017; root:xnu-4570.30.85~18/RELEASE_ARM64_T8015
12.1.1	Darwin Kernel Version 18.2.0: Mon Nov 12 20:32:01 PST 2018; root:xnu-4903.232.2~1/RELEASE ARM64 T8020
13.3	Darwin Kernel Version 19.2.0: Mon Nov 4 17:44:49 PST 2019; root:xnu-6153.60.66~39/RELEASE_ARM64_T8010
14.3 beta	Darwin Kernel Version 20.2.0: Sun Nov 1 23:50:23 PST 2020; root:xnu-7195.60.63~22/RELEASE_ARM64_T8015
15.2 beta 3	Darwin Kernel Version 21.2.0: Thu Nov 11 02:37:21 PST 2021; root:xnu-8019.60.69~8/RELEASE_ARM64_T8110

V · T · E

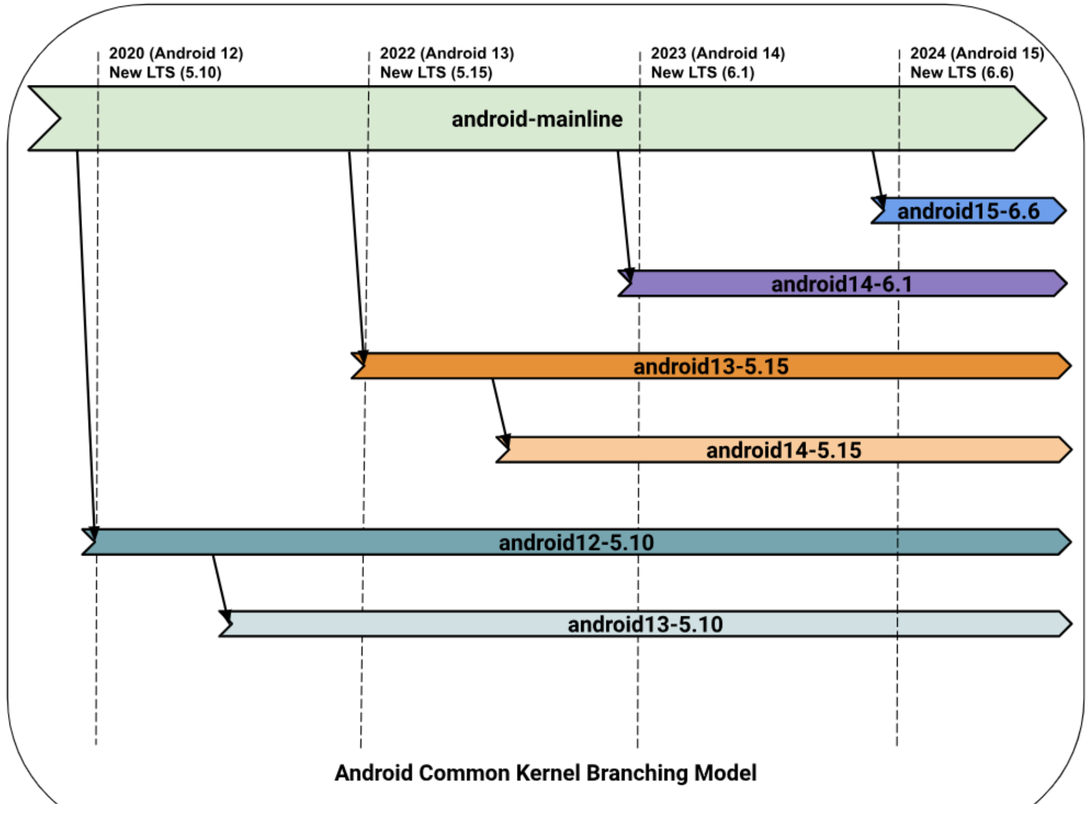
Supported iOS versions on the iPhone

iPhone models	iPhoneOS version			iOS version														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
iPhone 6s[b]	—	—	—	—	—	—	—	—	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗

EOL of LTS Kernel (kernel.org ≠ ACK)

Longterm release kernels

Version	Maintainer	Released	Projected EOL
6.6	Greg Kroah-Hartman & Sasha Levin	2023-10-29	Dec, 2026
6.1	Greg Kroah-Hartman & Sasha Levin	2022-12-11	Dec, 2026
5.15	Greg Kroah-Hartman & Sasha Levin	2021-10-31	Dec, 2026
5.10	Greg Kroah-Hartman & Sasha Levin	2020-12-13	Dec, 2026
5.4	Greg Kroah-Hartman & Sasha Levin	2019-11-24	Dec, 2025
4.19	Greg Kroah-Hartman & Sasha Levin	2018-10-22	Dec, 2024



Android Common Kernel Branching Model

ACK branch	Launch date	Support lifetime (years)	EOL
android-4.19-stable	2018-10-22	6	2025-01-01
android11-5.4	2019-11-24	6	2026-01-01
android12-5.4	2019-11-24	6	2026-01-01
android12-5.10	2020-12-13	6	2027-07-01
android13-5.10	2020-12-13	6	2027-07-01
android13-5.15	2021-10-31	6	2028-07-01
android14-5.15	2021-10-31	6	2028-07-01
android14-6.1	2022-12-11	6	2029-07-01
android15-6.6	2023-10-29	4	2028-07-01

There is no such thing as a free lunch

MOBILE → ANDROID OS

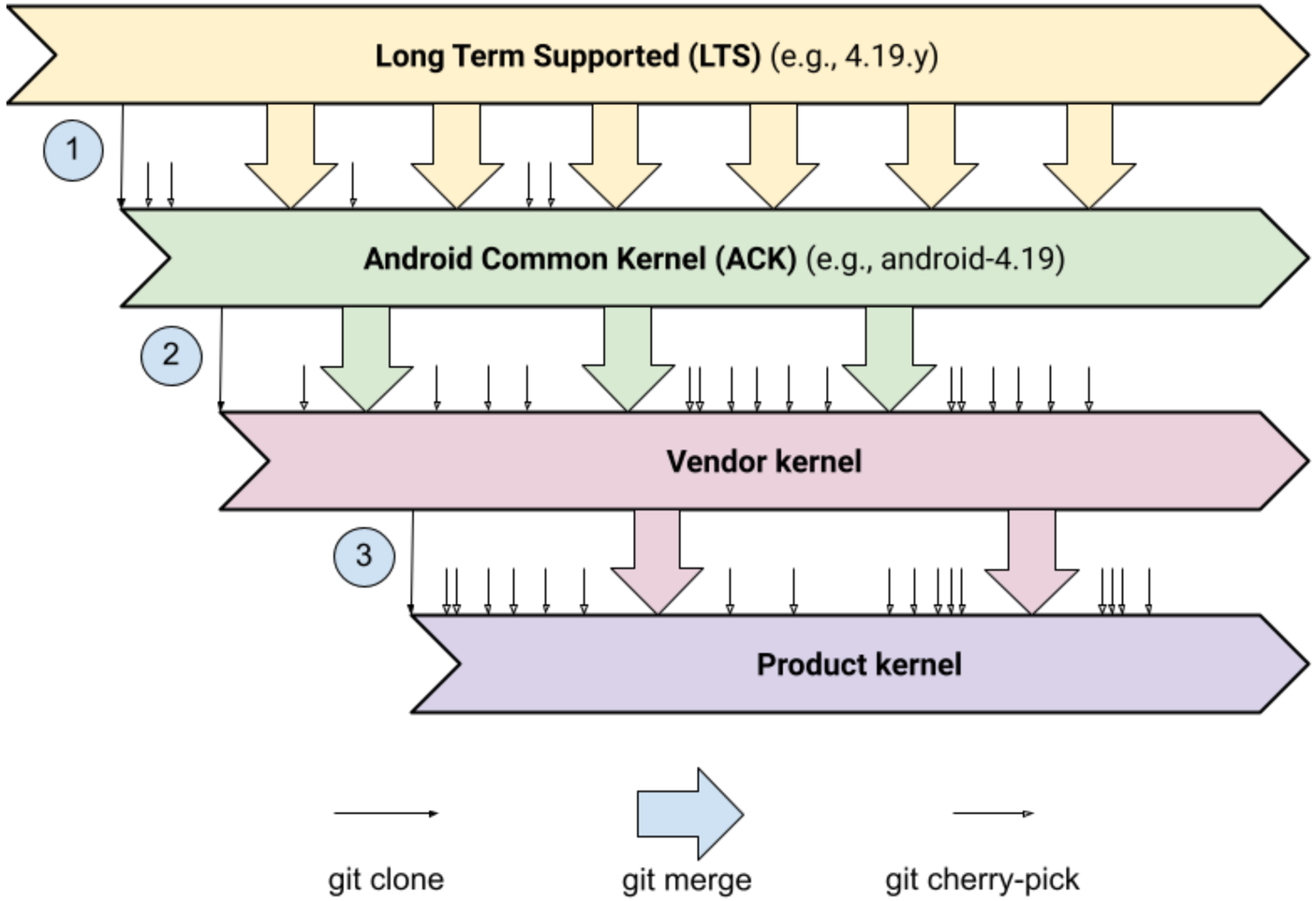
Google extends Linux kernel support to keep Android devices secure for longer

After Linux reduced LTS releases from 6 years to 2, Google has committed to supporting its forks for 4 years.

By Mishaal Rahman • July 8, 2024

- But LTS Kernel’s EOL cannot cover device lifespan.
- 6-year support lifetime of stable kernels are not long enough for modern devices.
- So, Kernel Uprev (major version upgrade) is essential.

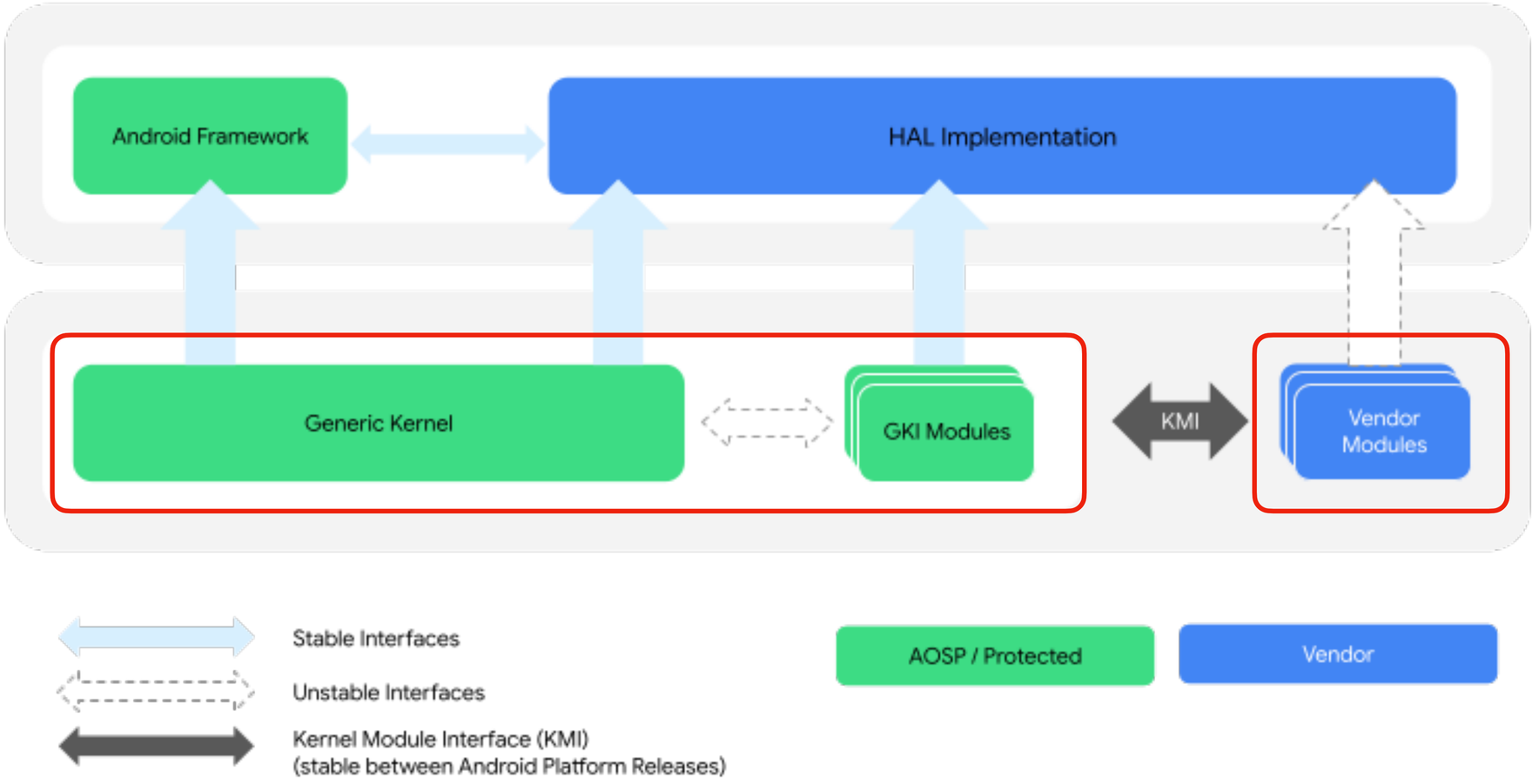
Before GKI (Generic Kernel Image)



Cost of Kernel Fragmentation

- Security updates are labor intensive
- Difficult to merge long-term stable updates
- Inhibits Android Platform release upgrades
- Difficult to contribute kernel changes back to upstream Linux

GKI (Generic Kernel Image)



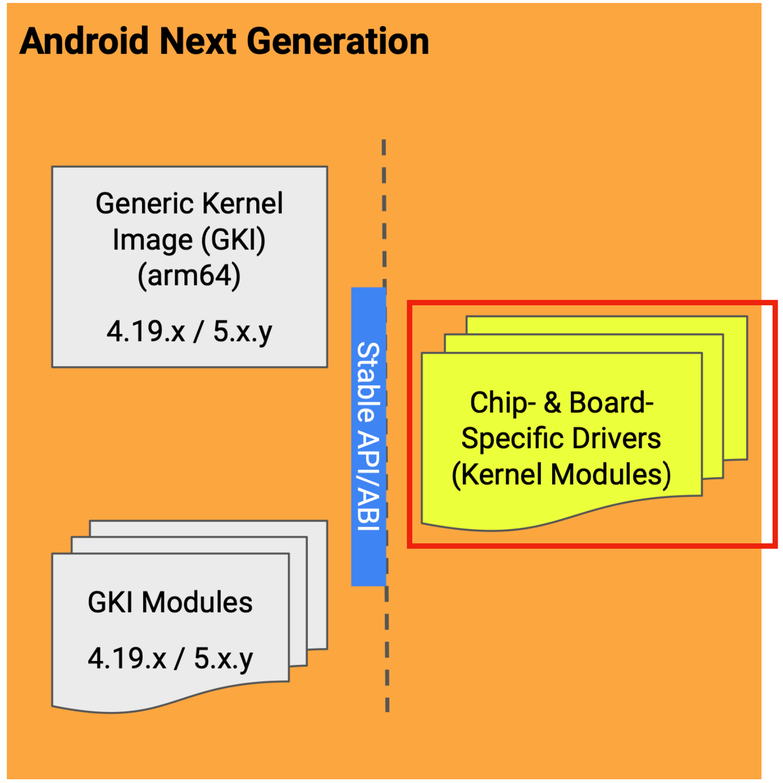
Branches | **Configuration** | **Toolchain** | **Scope**

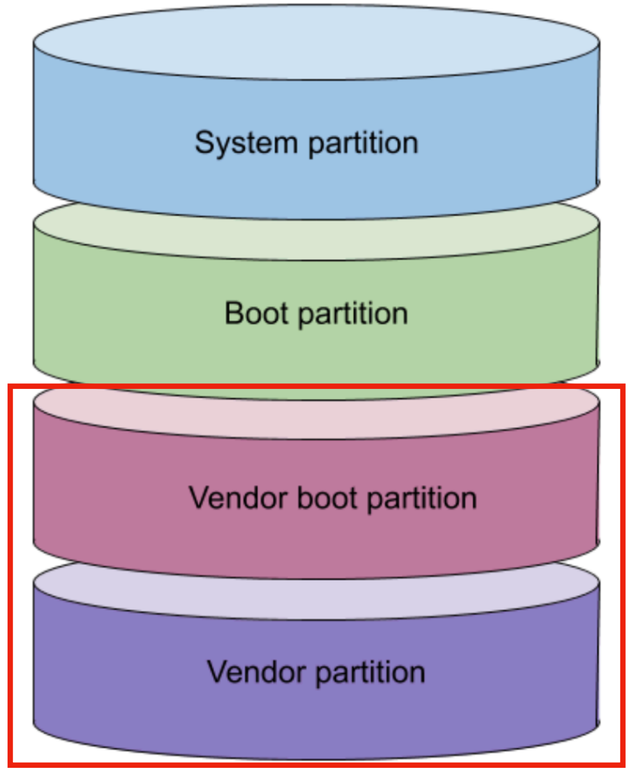
The necessity of module build

Vendor modules

A vendor module is delivered by partners to implement SoC and device-specific capabilities. Any existing kernel module that isn't delivered as part of the GKI kernel can be delivered as a vendor module.

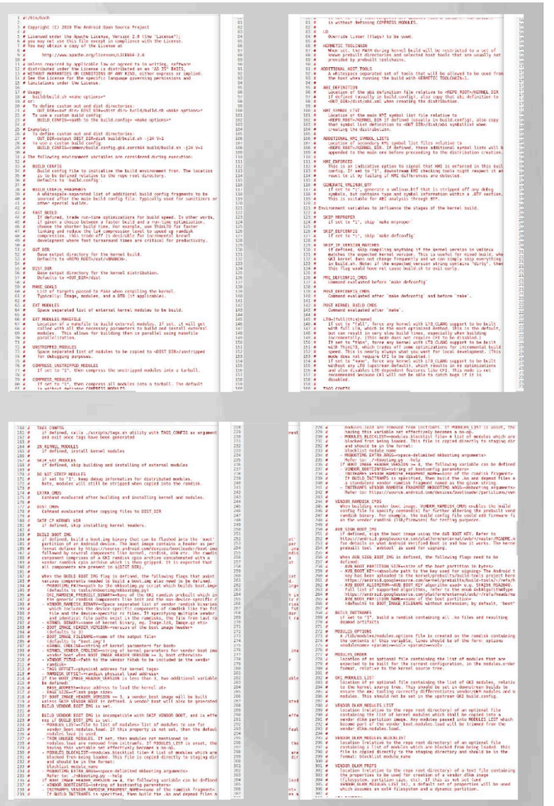
Since one of the primary goals of the GKI project is to minimize hardware-specific code in the core kernel, vendors can expect that the GKI kernel won't include modules that are clearly managing their own hardware. For example, vendor





Android Kernel build (deprecated way)

- build.sh
  - lots of control via env variables
  - ever-growing shell script collection
  - difficult to maintain
  - hermeticity / reproducibility as hack
  - limited parallelism



What is Bazel ?

Bazel (/ˈbeɪzəl/[3]) is a free and open-source software tool used for the automation of building and testing software.[2] Google uses the build tool *Blaze* internally[4] and released an open-source port of the Blaze tool as Bazel, named as an anagram of Blaze.[5] Bazel was first released in March 2015 and entered beta by September 2015.[6] Version 1.0 was released in October 2019.[7]

Similar to build tools like Make, Apache Ant, and Apache Maven,[2][5] Bazel builds software applications from source code using rules. Rules and macros are created in the Starlark language (previously called Skylark),[8] a dialect of Python.[5] There are built-in rules for building software written in Java, Kotlin, Scala, C, C++, Go, Python, Rust, JavaScript, Objective-C, and bash scripts.[5][6] Bazel can produce software application packages suitable for deployment for the Android and iOS operating systems.[9]

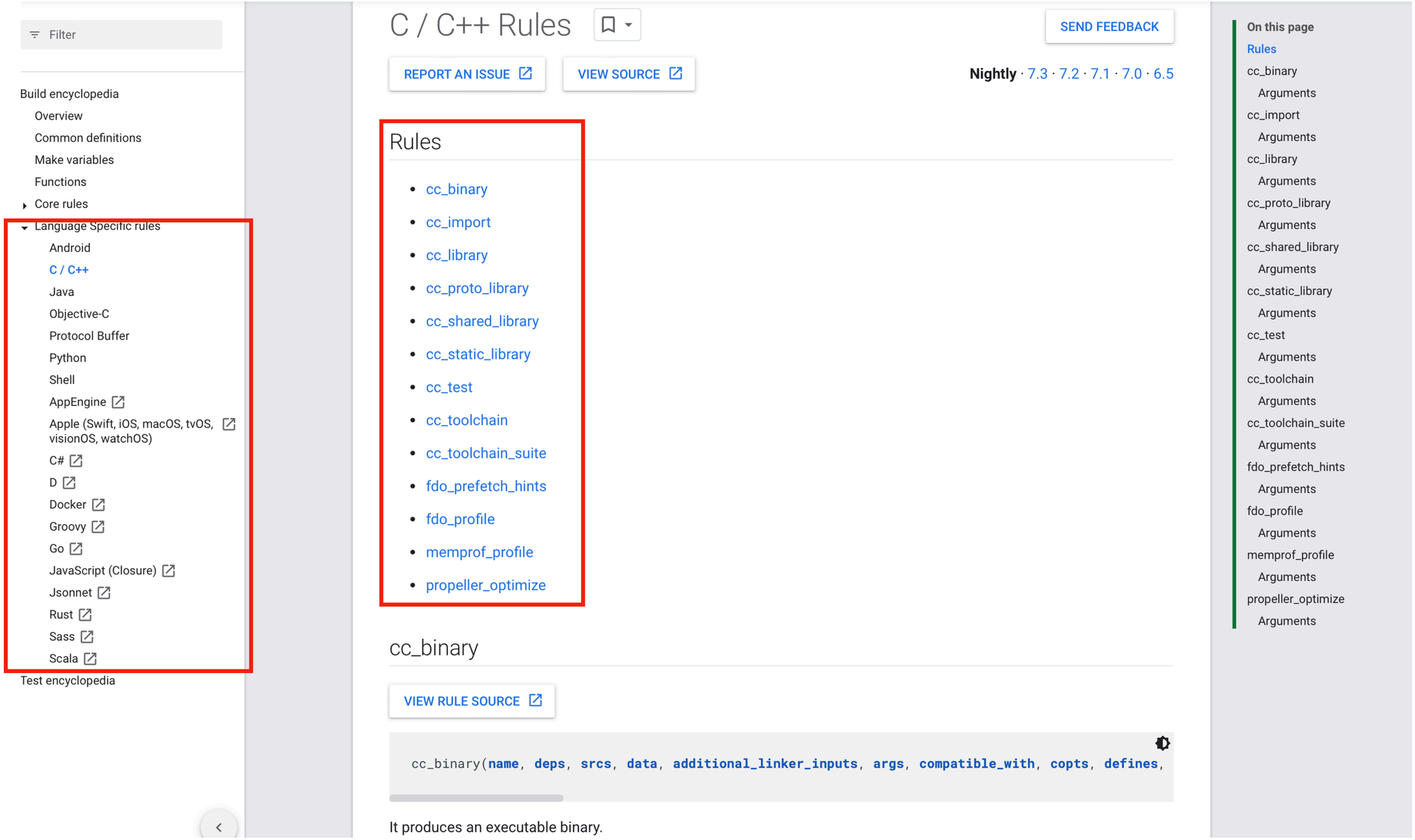
Rationale [edit]

One of Bazel's main purposes is to establish a build system in which the inputs and outputs of build targets are fully specified.

Bazel

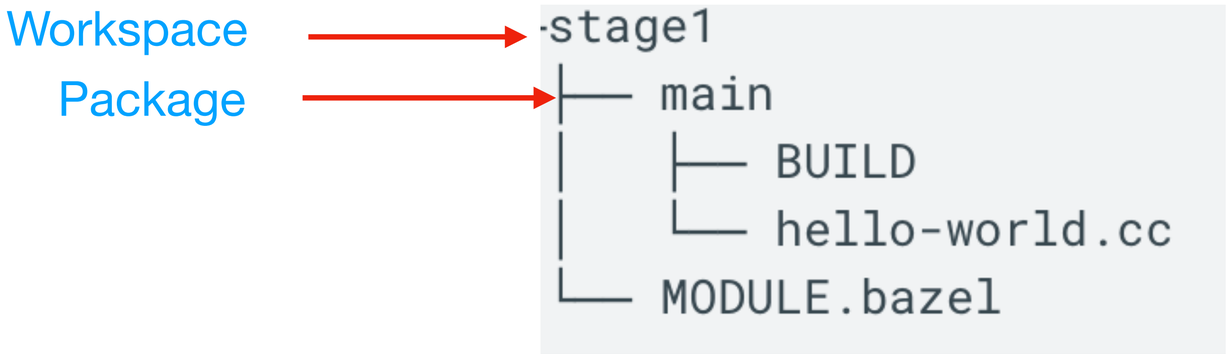
Developer(s)	Google
Initial release	March 2015; 9 years ago
Stable release	7.2.1 / 25 June 2024; 55 days ago[1]
Repository	github.com/bazelbuild/bazel
Written in	Java[2]
Operating system	Cross-platform
License	Apache License 2.0
Website	bazel.build

Languages & Rules on Bazel



Bazel 101

- A workspace is a directory that holds your project's source files and Bazel's build outputs.
- The MODULE.bazel file, which identifies the directory and its contents as a Bazel workspace and lives at the root of the project's directory structure.
- One or more BUILD(or BUILD.bazel) files, which tell Bazel how to build different parts of the project. A directory within the workspace that contains a BUILD file is a package.
- Each instance of a build rule in the BUILD file is called a target



Hello World using Bazel

< build & run >

```
$ bazel build //main:hello-world
$ bazel-bin/main/hello-world
```

< single source>

```
stage1
├── main
│   ├── BUILD
│   └── hello-world.cc
└── MODULE.bazel
```

```
cc_binary(
    name = "hello-world",
    srcs = ["hello-world.cc"],
)
```

< multi source>

```
stage2
├── main
│   ├── BUILD
│   ├── hello-world.cc
│   ├── hello-greet.cc
│   └── hello-greet.h
└── MODULE.bazel
```

```
cc_library(
    name = "hello-greet",
    srcs = ["hello-greet.cc"],
    hdrs = ["hello-greet.h"],
)

cc_binary(
    name = "hello-world",
    srcs = ["hello-world.cc"],
    deps = [
        ":hello-greet",
    ],
)
```

< multi package>

```
stage3
├── main
│   ├── BUILD
│   ├── hello-world.cc
│   ├── hello-greet.cc
│   └── hello-greet.h
├── lib
│   ├── BUILD
│   ├── hello-time.cc
│   └── hello-time.h
└── MODULE.bazel
```

Take a look at the `lib/BUILD` file:

```
cc_library(
    name = "hello-time",
    srcs = ["hello-time.cc"],
    hdrs = ["hello-time.h"],
    visibility = ["//main:__pkg__"],
)
```

And at the `main/BUILD` file:

```
cc_library(
    name = "hello-greet",
    srcs = ["hello-greet.cc"],
    hdrs = ["hello-greet.h"],
)

cc_binary(
    name = "hello-world",
    srcs = ["hello-world.cc"],
    deps = [
        ":hello-greet",
        "//lib:hello-time",
    ],
)
```

Android Kernel build with Bazel (a.k.a kleaf)

```
$ BUILD_CONFIG=common/build.config.gki.aarch64 build/build.sh
```

⬇️

```
$ bazel build //common:kernel_aarch64
```

<BUILD.bazel>

```
load("//build/kernel/kleaf:kernel.bzl",
     "kernel_build", "kernel_module")

kernel_build(
    name = "kernel",
    outs = ["vmlinux"],
    build_config = "common/build.config.gki.aarch64",
    srcs = glob(["**"]),
)

kernel_module(
    name = "nfc",
    srcs = glob(["**"]),
    outs = ["nfc.ko",],
    kernel_build = "//common:kernel",
)
```

kernel_build

Defines a kernel build target with all dependent targets.

It uses a `build_config` to construct a deterministic build environment (e.g. `common/build.config.gki.aarch64`). The kernel sources need to be declared via srcs (using a `glob()`). outs declares the output files that are surviving the build. The effective output file names will be `$(name)/$(output_file)`. Any other artifact is not guaranteed to be accessible after the rule has run. The default `toolchain_version` is defined with the value in `common/build.config.constants`, but can be overriden.

A few additional labels are generated. For example, if name is `"kernel_aarch64"`:

- `kernel_aarch64_uapi_headers` provides the UAPI kernel headers.
- `kernel_aarch64_headers` provides the kernel headers.

PARAMETERS

Name	Description	Default Value
name	The final kernel target name, e.g. `"kernel_aarch64"`.	none
build_config	Label of the build.config file, e.g. `"build.config.gki.aarch64"`.	none
outs	The expected output files.	none

kernel_module

Generates a rule that builds an external kernel module.

PARAMETERS

Name	Description	Default Value
name	Name of this kernel module.	none
kernel_build	Label referring to the kernel_build module.	none
outs	The expected output files. If unspecified or value is `None`, it is `["{name}.ko"]` by default.	`None`

Android Kernel build with Bazel (a.k.a kleaf)

```c
// my_mod.c

#include <linux/module.h>



MODULE_DESCRIPTION("A demo module");
MODULE_LICENSE("GPL v2");

void print_from_my_mod(void) {
  printk(KERN_INFO "Hello");
};

EXPORT_SYMBOL_GPL(print_from_my_mod);
```

```c
// my_other_mod.c

#include <linux/module.h>

#include "my_mod.h"

MODULE_DESCRIPTION("Another demo module");
MODULE_LICENSE("GPL v2");

void print_something(void) {
    print_from_my_mod();
};
```

No Kconfig, No Makefile

```
# BUILD.bazel

load("//build/kernel/kleaf:kernel.bzl", "ddk_module")

ddk_module(
    name = "my_mod",
    srcs = ["my_mod.c",],
    out = "my_mod.ko",
    hdrs = ["my_mod.h"],
    kernel_build = "//common:kernel",
    deps = ["//common:all_headers"],
)

ddk_module(
    name = "my_other_mod",
    srcs = ["my_other_mod.c",],
    out = "my_other_mod.ko",
    kernel_build = "//common:kernel",
    deps = [
        ":my_mod",
        "//common:all_headers",
    ],
)
```

TL;DR

- EU eco design regulation
- Kernel Uprev is essential for device longevity (Pixel 1 vs iPhone 6s)
- GKI
- Module build
- Bazel (Kleaf)

Question ?

References

- https://chatgpt.com/
- https://en.wikipedia.org/wiki/Google_Pixel
- https://android.googlesource.com/kernel/msm/+/refs/heads/android-msm-marlin-3.18-pie-qpr2
- https://en.wikipedia.org/wiki/IPhone
- https://en.wikipedia.org/wiki/IOS_version_history
- https://theapplewiki.com/wiki/Kernel#iOS/iPadOS
- https://www.kernel.org/category/releases.html
- https://source.android.com/docs/core/architecture/kernel/android-common#common-kernel-hierarchy
- https://source.android.com/docs/core/architecture/kernel/android-common#support-lifetimes
- https://www.androidauthority.com/google-extends-linux-support-3457871/
- https://source.android.com/docs/core/architecture/kernel/generic-kernel-image
- https://source.android.com/docs/core/architecture/kernel
- https://lpc.events/event/7/contributions/792/attachments/519/931/Update_on_GKI_KMI_enforcement_tools_1.pdf
- https://en.wikipedia.org/wiki/Bazel_(software)
- https://bazel.build/start/cpp
- https://bazel.build/reference/be/c-cpp
- https://lpc.events/event/16/contributions/1337/attachments/968/1891/LPC 2022 - Hermetic Builds with Bazel.pdf
- https://android.googlesource.com/kernel/build/+/refs/heads/main/kleaf/docs/api_reference/kernel.md#kernel_module
- https://lpc.events/event/17/contributions/1441/attachments/1163/2406/Simplified Android Kernel Driver Development with DDK v2.pdf