Low-Level API

Recommendation from prpl

Version 1.3

Revision History

Version	Date	Author	Description
0.5	2019.10.20	John Crispin (prpl) & Wouter Cloetens (Liberty Global)	v1 APIs
0.6	2019.10.28	Mirko Lindner (SmartRG)	Surrounding texts
1.0	2019-11-24	David Barr (Intel)	Approved text by the Prpl Technical Steering Committee
1.1	2020-01-29	Wouter Cloetens (Deutsche Telekom)	Cosmetic changes and addition of the 2017 approved WLAN API text. Fixed switchdev link.
1.2	2020-05-05	Wouter Cloetens (Deutsche Telekom)	Wireless: updated kernel documentation link. Appendix 1: updated linux-wireless document link.
1.3	2022-02-14	Wouter Cloetens (Deutsche Telekom)	Updated kernel documentation links. Removed John Crispin from contact list. Reorganised by sections. Replaced button recommendation with gpio-keys.

			RFS/XPS: marked as optional. Elaborated on Ethernet modeling, added ethtool. Deprecated swdev. Added eMMC. Replaced "Features not part of LLAPI" with "Features under review." Added ePON to the list of currently out of scope interfaces. Rewrote packet offload, adding IPsec, and QoS sections. Added copyright preamble section.

Content

1. Important notices, IPR statement, disclaimer and Copyright

This chapter contains important information about PRPL and this document (hereinafter 'This PRPL Document').

1.1 About PRPL

The prpl Foundation (PRPL) is a not-for-profit organization which publishes documents including, but not limited to, Requirements, Specifications, Recommendations, API application programming interfaces, and Test Plans.

1.2 This may not be the latest version of This PRPL Document

This PRPL Document is the output of the Working Groups of the PRPL and its members as of the date of publication. Readers of This PRPL Document should be aware that it can be revised, edited or have its status changed according to the PRPL working procedures.

1.3 There is no warranty provided with This PRPL Document

The services, the content and the information in This PRPL Document are provided on an "as is" basis. PRPL, to the fullest extent permitted by law, disclaims all warranties, whether express, implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third-parties rights and fitness for a particular purpose. PRPL, its affiliates and licensors make no representations or warranties about the accuracy, completeness, security or timeliness of the content or information provided in the PRPL Document. No information obtained via the PRPL Document shall create any warranty not expressly stated by PRPL in these terms and conditions.

1.4 Exclusion of Liability

Any person holding a copyright in This PRPL Document, or any portion thereof, disclaims to the fullest extent permitted by law (a) any liability (including direct, indirect, special, or consequential damages under any legal theory) arising from or related to the use of or reliance upon This PRPL Document; and (b) any obligation to update or correct this technical report.

1.5 THIS PRPL DOCUMENT IS NOT BINDING ON PRPL NOR ITS MEMBER COMPANIES

This PRPL Document, though formally approved by the PRPL member companies, is not binding in any way upon the PRPL members.

1.6 INTELLECTUAL PROPERTY RIGHTS

Patents essential or potentially essential to the implementation of features described in This PRPL Document may have been declared in conformance to the PRPL IP Policy (available at the PRPL website: www.prplFoundation.org).

1.7 COPYRIGHT PROVISIONS

This PRPL Document is copyrighted by PRPL, and all rights are reserved. The contents of This PRPL Document are protected by the copyrights of PRPL or the copyrights of third-parties that are used by agreement. Trademarks and copyrights mentioned in This PRPL Document are the property of their respective owners. The content of This PRPL Document may only be reproduced, distributed, modified, framed, cached, adapted or linked to, or made available in any form by any means, or incorporated into or used in any information storage and retrieval system, **with the prior written permission of PRPL or the applicable third-party copyright owner**. Such written permission is **not** however required under the conditions specified in Section 1.7.1 and Section 1.7.2:

1.7.1 INCORPORATING PRPL DOCUMENTS IN WHOLE OR PART WITHIN DOCUMENTS RELATED TO COMMERCIAL TENDERS

Any or all section(s) of PRPL Documents may be incorporated into Commercial Tenders (RFP, RFT, RFQ, ITT, etc.) by PRPL and non-PRPL members under the following conditions:

1. The PRPL Requirements numbers, where applicable, must not be changed from those within the PRPL Documents;
2. A prominent acknowledgement of the PRPL must be provided within the Commercial document identifying any and all PRPL Documents referenced and giving the web address of the PRPL;
3. The Commercial Tender must identify which of its section(s) include material taken from PRPL Documents and must identify each PRPL Document used, and the relevant PRPL Section Numbers; and,
4. The Commercial Tender must refer to the copyright provisions of PRPL Documents and must state that the sections taken from PRPL Documents are subject to copyright by PRPL and/or applicable third parties.

1.7.2 COPYING THIS PRPL DOCUMENT IN ITS ENTIRETY

This PRPL Document may be electronically copied, reproduced, distributed, linked to, or made available by other means, or incorporated into or used in any information storage and retrieval system, but **only in its original, unaltered PDF format**, and with its original PRPL title and file name unaltered. It may not be modified without the advanced written permission of the PRPL.

2. Introduction

This document identifies low-level APIs recommended by prpl to promote harmonization and convergence among SW platforms.

The recommended APIs should be supported by BSPs in order to best leverage the efforts from open-source software communities.

Harmonization and convergence is needed because the current landscape employs disparate abstraction layers from numerous different platform providers.

This proliferation of different abstraction layers has made porting efforts more difficult, harder to maintain, and hence unscalable to large numbers of CPE platforms.

Prpl therefore recommends instead the best practice of relying on standard kernel interfaces, wherever possible.

This recommendation lists those APIs corresponding to various BSP subsystems.

Some of the APIs are already proven and mature, and hence implementable now.

Others remain works-in progress, suitable for prototype development and verification.

While others are forward-looking toward the future.

Thus, prpl expects these recommendations to be updated over time—see https://prplfoundation.org/documents/.

3. Background

Introduction to the Low-Level API

In a world of growing complexity and a need by ISPs to launch new devices at an ever-faster pace, the industry is looking for ways to reduce product development times in an effort to cut time-to-market as much as possible.

Open API efforts such as prpl's High-Level API are designed to drastically reduce the effort and thereby time required to integrate an ISP's services on a new device and software platform.

However, a big driver in time-to-market is the effort to port any software stack to a new chipset, especially if this chipset comes from an SoC vendor the software integrator is unfamiliar with. Integrating new board support packages or SoC SDKs can be a lengthy and difficult process. A miscalculation or unexpected finding can threaten a project's overall success.

Historically, chipset SDKs are almost complete CPE stacks which were designed to help the hardware vendor to launch the product as quickly as possible. With the increase of services and customization required by each ISP, software vendors have begun to replace large parts of the SoC's SDK, some even only use the bare drivers. As long as the underlying architecture remains relatively unchanged, this is a valid and promising approach.

However, in addition to more services ISPs have moved to also require different chipsets and solutions from different SoC vendors to be used in their CPEs. Some do so because of procurement and continuity strategies, others simply because they deployed technologies that require a specific chipset to be used. In either case, the pressure to find ways to speed up product development increases even further.

Software vendors and hardware manufacturers are now faced with ISP customers who are searching for partners who can support more customization and services on a greater number of devices using the largest variety of chipset vendors at the fastest pace possible.

To do so, industry representatives in prpl have agreed to establish a new standard integration boundary for software stacks on top of SoC SDKs. Instead of every software integrator integrating every driver and SDK feature, the SoC vendors are being asked to consolidate their APIs and offer all core features through open, publicly available and peer-reviewed kernel APIs - the *prpl Low-Level API*. Standardizing these interfaces will dramatically reduce the time it takes a software vendor to integrate a chipset maker's SDK and also result in accelerated innovation throughout the whole industry.

Of course, an Open API effort can never capture 100% of all functionality and SoC vendors will want to continue to innovate and differentiate. Therefore it is important to note that the *Low-Level API* is not meant to be the only API between CPE stack and SoC SDK, but it is meant to be the library of how standard features are to be exposed and made available to the higher layers.

Way of working

In order to make this transition to a harmonized API an evolution every company can participate in and benefit from rather than disruption, the industry representatives have designed a multi-step approach.

The first step is the creation of API proposals based on existing kernel APIs. Once ratified by prpl, representatives will engage with the relevant chipset vendors to seek their agreement and support for the proposed interfaces.

As part of the engagement, prpl will work with the chipset vendors to establish timelines for when their SDK will support which of the proposed interfaces

The final step is the creation of a validation tool that will help SoC vendors prove their conformity and help software integrators when adopting new chipsets and their SDKs.

This process will be repeated for each release of API candidates and may be accompanied by further naming and numbering schemes.

Purpose of this document

This document describes the first set of API candidates the prpl low-level API working group has identified. It will serve as the baseline for the conversations with chipset vendors as described above. As such this document does not describe any formal API or requirement but is a working document.

Intended audience

This document is meant to be used by the technical teams at the SoC vendors to assess the direction of the Low-Level API and compare it with their own use of kernel APIs.

1. API

Peripheral Interfaces

Non-networking peripheral interfaces.

GPIO

Feature	Kernel Subsystem	Kernel API	Dependencies
GPIO	GPIO	sysfs/gpiolib	None
https://www.kernel.org/doc/html/latest/driver-api/gpio/index.html			

GPIO is a basic requirement for various other features and subsystems. The Vendor kernel shall provide support via the default kernel interface. If the silicon supports GPIO interrupts, these shall be supported using the kernel's IRQ subsystem.

Button

Feature	Kernel Subsystem	Kernel API	Dependencies
Button	gpio-keys	input	gpiolib, gpio, devicetree
Input subsystem: https://www.kernel.org/doc/html/latest/driver-api/input.html Buttons connected to GPIO must be modelled using gpio-keys • with interrupt support: gpio_keys.c			

- without: gpio_keys_polled.c

Modelling through DeviceTree: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/devicetree/bindings/input/gpio-keys.yaml

Buttons not connected to GPIO interfaces may be exposed through the input subsystem in a different fashion.

The aim is to align on common keycodes for typically present pushbuttons on gateways and access points. A (WPS) pushbutton must use KEY_WPS_BUTTON. Others are to be defined.

PWM

Feature	Kernel Subsystem	Kernel API	Dependencies
PWM	PWM	sysfs	None
https://www.kernel.org/doc/html/latest/driver-api/pwm.html			

PWM is a basic requirement for controlling a LEDs brightness. If the silicon has a PWM block, it shall be supported by the kernel's PWM layer.

LED

Feature	Kernel Subsystem	Kernel API	Dependencies
LED	LED	sysfs/leds	GPIO/PWM

https://www.kernel.org/doc/html/latest/leds/index.html

LEDs shall be supported using the kernel's LED subsystem. This can be done via a custom, the leds-gpio or the leds-pwm driver. All LEDs shall follow the kernel's naming scheme.

I2C

Feature	Kernel Subsystem	Kernel API	Dependencies
I^2C	I2C	ioctl/i2cdev	(GPIO)
https://www.kernel.org/doc/html/latest/i2c/index.html https://www.kernel.org/doc/html/latest/driver-api/i2c.html			

I^2C busses shall be supported using the kernel's I^2C subsystem. This can be done via a custom or the i2c-gpio driver.

SPI

Feature	Kernel Subsystem	Kernel API	Dependencies
SPI	SPI	ioctl/spidev	(GPIO)
https://www.kernel.org/doc/html/latest/driver-api/spi.html			

SPI busses shall be supported using the kernel's SPI subsystem. This can be done via a custom or the spi-gpio driver.

Sensors

Feature	Kernel Subsystem	Kernel API	Dependencies
Sensors	IIO	sysfs	(GPIO/I2C/SPI)
https://www.kernel.org/doc/html/latest/driver-api/iio/index.html			

Sensors supported/required by the solution shall be supported using the kernel's IIO subsystem.

CPU

Frequency and Voltage Scaling

Feature	Kernel Subsystem	Kernel API	Dependencies
Frequency Scaling	cpu-freq	sysfs	None
https://www.kernel.org/doc/html/latest/cpu-freq/index.html			

All dynamic frequency and voltage scaling of the SoC cores has to happen via the kernel's cpu-freq subsystem. Governors shall be supported to control behaviour.

CPU Idle

Feature	Kernel Subsystem	Kernel API	Dependencies
CPUIdle	cpuidle	sysfs	None
https://www.kernel.org/doc/html/latest/admin-guide/pm/cpuidle.html https://www.kernel.org/doc/html/latest/driver-api/pm/cpuidle.html			

As soon as the OS enters the idle loop for a CPU core or thread, SoC-specific behaviour and any special sleep modes shall be integrated in the kernel's CPUIdle subsystem. Governors shall be supported to control behaviour.

CPU Cooling

Feature	Kernel Subsystem	Kernel API	Dependencies
CPU Cooling	thermal	sysfs	None
https://www.kernel.org/doc/html/latest/driver-api/thermal/cpu-cooling-api.html			

If a CPU cooling method is supported, it must be registered with the CPU cooling API, to allow control by thermal governors.

Generic Thermal Framework

Feature	Kernel Subsystem	Kernel API	Dependencies
Thermal Zone sysfs	thermal	sysfs	None
https://www.kernel.org/doc/html/latest/driver-api/thermal/sysfs-api.html			

The generic thermal sysfs provides a set of interfaces for thermal zone devices (sensors) and thermal cooling devices (fan, CPU). Any supporting hardware must register with these.

Packet Offload

A gradual transition is planned from proprietary integrations of packet offload mechanisms, by hooking into the Linux networking stack, to a unified open source flow identification and software packet offload mechanism, with clearly defined hooks for hardware solutions.

Functional requirements:

It must be possible to:

1. control at what point exactly offload of a flow starts
2. interrupt offload of a flow
3. configure the number of packets of a new flow that go through the slowpath (Linux networking stack) before offload starts
4. offload a multicast flow as of the first packet, specifying the ingress interface and egress interfaces
5. modify the list of multicast egress interfaces (add/remove) of an offloaded multicast flow
6. stop offload of a multicast flow, taking into account that there is a delay between sending a leave request and when the last packet arrives
7. read the following per unicast IP flow metrics, regardless of whether the flow is handled through the Linux networking stack or whether it is offloaded in software or hardware: rx/tx packets, rx/tx bytes
8. query the state of a unicast IP flow, and be notified in real time of state changes
9. read the following multicast IP flow metrics, regardless of whether the flow is handled through the Linux networking stack or whether it is offloaded in software or hardware: rx/tx packets, rx/tx bytes

10. A flow's transition from slowpath to fastpath SHOULD not cause packets to be delivered out of sequence.

Phase 1: correct interaction with Linux

1. When SKB mark bit 0x20 is set, neither software nor hardware packet offload of a flow of which the packet may be part may be started. In phase 1, it is acceptable for static configuration (e.g. netfilter rules) to translate this to a proprietary mechanism, such as a different SKB mark or a netfilter target.
2. When the conntrack flow entry is deleted (possible from user space through the netlink interface), flow offload must stop.
3. A proprietary interface to configure the number of packets statically, typically at boot time, is acceptable.
4. A proprietary multicast control mechanism is acceptable.
5. A proprietary multicast control mechanism is acceptable.
6. A proprietary multicast control mechanism is acceptable.
7. Conntrack metrics must be updated, if not in real-time, then at regular intervals. When the conntrack flow is terminated, the metrics must be accurate.
8. Conntrack integration is required.
9. A proprietary multicast management interface is acceptable.

Phase 2: standardised user space and kernel interfaces

1. The SKB mark bit must be sufficient information to postpone offload.
2. No change.
3. Interface to be defined.
4. Interface to be defined.
5. Interface to be defined.
6. Interface to be defined.
7. No change.

8. No change.
9. Interface to be defined.

Phase 3: Flowtable offload

Feature	Kernel Subsystem	Kernel API	Dependencies
Flowtable offload	net	nftables	V5.7
https://www.kernel.org/doc/html/latest/networking/nf_flowtable.html			

Silicon supporting HW accelerated networking shall hook its hardware offload capabilities into the flowtable offload infrastructure.

RFS/XPS

Feature	Kernel Subsystem	Kernel API	Dependencies
RFS/XPS	net	sysfs	None
https://www.kernel.org/doc/html/latest/networking/scaling.html			

In the rare case that the Silicon supports frame steering in HW, it shall be hooked into the kernel's standard configuration interface.
This interface is optional.

IPsec offload

Feature	Kernel Subsystem	Kernel API	Dependencies
XFRM	net	sysfs	None
https://www.kernel.org/doc/html/latest/networking/xfrm_device.html			

Any hardware-based IPsec offload must integrate into the XFRM (network) device interface.

QoS

A gradual transition is planned from proprietary integrations of QoS queue configuration and classification mechanisms to a standard interface in the packet offload system.

It is recognized that silicon vendors have hardware-assisted QoS capabilities that do not correspond to the software-based Linux kernel *traffic control* schedulers.

Phase 1: proprietary API

It is acceptable that the vendor provides a proprietary user space API for queue and scheduler configuration purposes.

Per-packet classification is typically performed by netfilter rules. These set the five least significant bits of the SKB mark of a packet to a queue number. It is acceptable that, at a later stage is netfilter packet processing, this is converted to a proprietary manner of classifying the packet, e.g. using a netfilter rule.

Phase 2: Linux tc

In this phase, the userspace-kernel interface is replaced with the netlink interface used by tc.

https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.qdisc.classful.html

Queue and scheduler configuration must occur through Linux tc. Proprietary queue disciplines with proprietary parameters may be defined.

Per-packet classification is typically performed by netfilter rules. These set the five least significant bits of the SKB mark of a packet to a queue number. Any low-level implementation must respect this SKB classification mark.

Phase 3: integration in flowtable

To be defined at a later time.

Wireless LAN

Feature	Kernel Subsystem	Kernel API	Dependencies
wireless LAN	net/wireless	cfg80211/nl80211	None
ll-api https://www.kernel.org/doc/html/latest/driver-api/80211/index.html			

Wireless drivers need to provide support for cfg80211/nl80211 in a manner that will allow an upstream hostapd/wpas to function properly.

See WLAN API for the full text of the recommendation.

Ethernet

Each Ethernet port shall be represented as an individual Ethernet Linux network interface. If a port is not represented in hardware by a dedicated MAC and PHY, then it shall be modelled as such.

Ethernet PHY

Feature	Kernel Subsystem	Kernel API	Dependencies
Ethernet PHY	PHY	netlink/ioctl	netlink: V5.6
https://www.kernel.org/doc/html/latest/networking/phy.html https://www.kernel.org/doc/html/latest/networking/ethtool-netlink.html			

The Ethernet switch device driver model (switchdev) is an in-kernel driver model for switch devices which offload the forwarding (data) plan.

DSA/switchdev

Feature	Kernel Subsystem	Kernel API	Dependencies
Ethernet switch	DSA/PHY/bridge	netlink/ioctl/sysfs	V4.9
https://www.kernel.org/doc/html/latest/networking/switchdev.html			

The Ethernet switch device driver model (switchdev) is an in-kernel driver model for switch devices which offload the forwarding (data) plane from the kernel. This must be supported.

Note that DSA-supported switches are implicitly supported by switchdev. DSA support by itself is not mandatory; only switchdev support is.

Multicast

MDB/mcast

Feature	Kernel Subsystem	Kernel API	Dependencies
MDB/mcast	net/bridge	MDB/notifier chain	SSM: v5.14+
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/net/bridge/br_mdb.c			

The kernel MDB should be kept in sync with the multicast memberships.

Cryptography

Hardware cryptography

Feature	Kernel Subsystem	Kernel API	Dependencies
HW crypto	crypto	in-kernel/cryptodev	None
https://www.kernel.org/doc/html/latest/crypto/index.html			

The kernel crypto API offers a rich set of cryptographic ciphers as well as other data transformation mechanisms and methods to invoke these. This document contains a description of the API and provides example code.

Hardware RNG

Feature	Kernel Subsystem	Kernel API	Dependencies
HW RNG	random, crypto	hw_random, random	None
https://www.kernel.org/doc/html/latest/admin-guide/hw_random.html https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/include/linux/hw_random.h			

Hardware-implemented true random number generators must be exposed via the crypto subsystem.

Storage

MTD

Feature	Kernel Subsystem	Kernel API	Dependencies
MTD	MTD	ioctl	None

The MTD subsystem provides support for NOR and NAND flash devices.

Vendor kernels may not provide their own bad block management layer but should rely on UBI to do the work.

MMC/SD/SDIO

Feature	Kernel Subsystem	Kernel API	Dependencies
eMMC	MMC	mmc/blkdev	None
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/mmc			

eMMC flash devices handle bad block management internally, and must be exposed as a plain block device, similar to a SATA hard disk.

Linux Containers

Container privileged

Feature	Kernel Subsystem	Kernel API	Dependencies
Container - privileged			V3.8+

Container unprivileged

Feature	Kernel Subsystem	Kernel API	Dependencies
Container - unprivileged			v4.19+

Features under review

- VLANs must be supported through Linux networking; to be documented
- DSL/G.fast, ATM/PTM
- VoIP endpoint interfaces for codec, SLIC/SLAC, TDM abstraction
- SFP cages
- GPON
- ePON
- Optical transceiver
- Cellular data, over USB, PCI or integrated
- DOCSIS

Contacts

The following table describes the primary contacts to ask about the low-level API.

Contact person	Email address
Wouter Cloetens	wouter.cloetens@prplfoundation.org

Appendix 1: WLAN API

Recommendations

The WLAN API recommendations are:

- cfg80211 driver for WLAN
- hostapd, wpa_supplicant, and iw support with WLAN management frames transmitted via nl80211

Scope

The primary focus is the control interface to radio and access point functionality, with STA (client) functionality as a lower priority. The data path is generally not a problem, as all known existing drivers model this as a Linux Ethernet device.

Availability of source code, and the license of, the device driver, microcode, and any firmware on an embedded offloaded processor, are out of scope.

The operating system scope is limited to Linux.

Reasoning

A lack of common interfaces for WLAN has a negative effect on all involved parties. Open source OpenWrt (or other open source router platform) developers are locked out from the use of officially supported drivers, which restricts which silicon solutions they can support. Router middleware vendors find themselves designing their own HAL's, and maintaining several implementations. WLAN is noted by middleware vendors as one of the most complex layers to maintain across chipset SDKs and even across versions of the same SDK.

While SDK users struggle with the lack of a common WLAN API, silicon vendors also maintain multiple interfaces. Those that support OpenWrt develop and maintain their own interface layer. Those that support RDK-B face the same effort with the RDK-B HAL. Those that support other stacks, e.g. their own proprietary one, maintain yet another.

All parties can benefit from a commonly agreed upon interface. More devices supported by OpenWrt benefits the project's growth. Silicon vendors would only need to support a single interface for all Linux-based devices. Middleware vendors will still need some higher level management software, but the responsibility of the maintenance of the layer between this and the common API will be moved firmly to the middleware vendor. In addition, only one such implementation is needed. For example, a single RDK-B HAL implementation would support all drivers, and therefore, the silicon vendors would not need to maintain a HAL implementation for their chips.

Recommended APIs

The prpl Carrier Interest Group evaluated the Linux WLAN space and formally recommends the creation and use of drivers based on cfg80211. Cfg80211 is actively maintained by the Linux kernel community and technically and functionally vastly superior to the older ioctl-based wireless interfaces. Additional background on the reason for the kernel community's move from ioctl interfaces to cfg80211/nl80211 can be found at:

https://wireless.wiki.kernel.org/_media/en/developers/documentation/control.pdf.

Members of prpl Foundation are strongly encouraged to work with the Linux Wireless community to add missing features into cfg80211, especially features which could be used across multiple vendor implementations.

The standard WLAN driver delivery should include the cfg80211 management API. In the event that the driver is part of a board support package (BSP) for a SoC, the included Linux kernel should have cfg80211 enabled, as should the WLAN driver. The

use of the cfg80211 by higher level management software, that may be included in the BSP, is recommended.

In addition to cfg80211 drivers, the Carrier Interest Group recommends the use of hostapd, wpa_supplicant and iw connected to the driver via nl80211. In particular, WLAN management frames should be transmitted between the driver and hostapd/wpa_supplicant/iw via nl80211.

Vendor-specific extensions should be limited to hardware or implementation specific functionality that is distinctly not present in cfg80211, and does not lend itself to an extension of cfg80211 either.

Examples are control of embedded firmware, debugging interfaces to the lower layers of the implementation, and special calibration logic. Any interfaces that must be used by an operational control interface should be standardized. The recommended interface for vendor-specific extensions is cfg80211 vendor commands.

References