Configstore HAL

Android O splits the monolithic Android OS into generic (system.img) and hardware-specific (vendor.img and odm.img) partitions. As a result of this change, conditional compilation must be removed from modules installed to the system partition and such modules must now determine the configuration of the system at runtime (and behave differently depending on that configuration).

The ConfigStore HAL provides a set of APIs for accessing read-only configuration items used to configure the Android framework. This page describes the design of ConfigStore HAL (and why system properties were not used for this purpose); other pages in this section detail the HAL interface (https://source.android.com/devices/architecture/configstore/interface.html), service implementation (https://source.android.com/devices/architecture/configstore/service.html), and client-side usage (https://source.android.com/devices/architecture/configstore/client.html), all using `surfaceflinger` as an example. For help with ConfigStore interface classes, see Adding Interface Classes & Items (https://source.android.com/devices/architecture/configstore/add-class-item.html).

Why not use system properties?

We considered using system properties but found several fundamental issues, including:

- **Length limits on values**. System properties have tight limits on the length of their values (92 bytes). In addition, as these limits have been directly exposed to Android apps as C macros, increasing the length can cause backwards-compatibility issues.
- **No type support**. All values are essentially strings, and APIs simply parse the string into an `int` or `bool`. Other compound data types (array, struct, etc.) should be encoded/decoded by the clients (e.g. "aaa,bbb,ccc" can be decoded as an array of three strings).
- **Overwrites**. Because read-only system properties are implemented as write-once properties, vendors/ODMs that want to override AOSP-defined read-only values must import their own read-only values prior to AOSP-defined read-only values, which in turn results in vendor-defined re-writable values being overridden by AOSP-defined values.
- **Address space requirements**. System properties take a relatively large amount of address space in each process. System properties are grouped in `prop_area` units with a fixed size of 128KB, all of which is allocated to a process address space even if only a single system property in it is being accessed. This can cause problems on 32-bit devices where address space is precious.

We attempted to overcome these limitations without sacrificing compatibility but continued to be concerned that system properties were not designed to support accessing read-only configuration items. Eventually we decided that system properties are better suited for sharing a few dynamically-updated items across all of Android in real time, and that a need existed for a new system dedicated to accessing read-only configuration items.

ConfigStore HAL design

The basic design is simple:

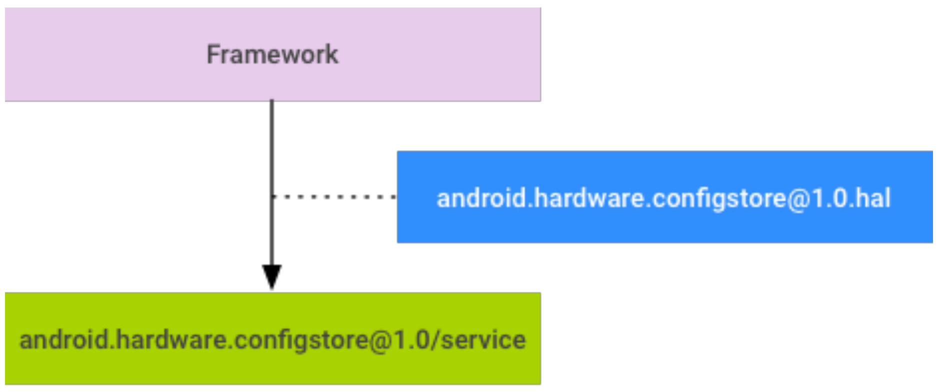
Framework
android.hardware.configstore@1.0.hal
android.hardware.configstore@1.0/service

Figure 1. ConfigStore HAL design

- Describe build flags (currently used for conditionally compiling the framework) in HIDL.
- Vendors and OEMs provide SoC and device-specific values for build flags by implementing the HAL service.
- Modify the framework to use the HAL service to find the value of a configuration item at runtime.

Configuration items currently referenced by the framework are included in a versioned HIDL package (`android.hardware.configstore@1.0`). Vendors and/or OEMs provide values to the configuration items by implementing interfaces in this package, and the framework uses the interfaces when it needs to get a value for a configuration item.

Security considerations

Build flags defined in the same interface are affected by same SELinux policy. If one or more build flags should have different SELinux policies, **they must be separated to another interface**. This can require major uprev of `android.hardware.configstore package` as the separated interfaces are no longer backwards-compatible.

Note: For details on Android 8.0 SELinux, see SELinux for Android 8.0 (https://source.android.com/security/selinux/images/SELinux_Treble.pdf).