Android 10.0 Sysprop as Api Introduction

WWW.UNISOC.COM
紫 光 展 锐 科 技

修改历史

版本号 Version	日期 Date	注释 Notes
V1.0	2019/09/23	初稿
V1.1	2019/12/26	适用平台信息新增UIS8581E\SL8541E
V1.2	2020/03/18	修改文档名称，更新文档格式，适用平台信息新增UIS7862。

文档信息

适用产品信息 Chip Platform	适用版本信息 OS Version	关键字 Keyword
SC9863A\SC9832E\SC7731E\T610\T618\UIS8581E\SL8541E\UIS7862	Android 10.0	Sysprop property

Contents

Contents

Introduction

Sysprop as api is a new feature on Android 10.0. As the literal meaning, In Sysprop as api rule, all access to property must by using the api, so why we need sysprop as api, What's wrong with the previous access method?

- Interfaces aren't explicitly defined
  - Hard to know exact dependencies
- Stability isn't guaranteed
  - Can be deleted/altered anytime
- No schema at code level
  - Keys are random magic strings (OH NO)
  - Values are stored as strings, regardless of type (Please..)

So, sysprop as api is coming.

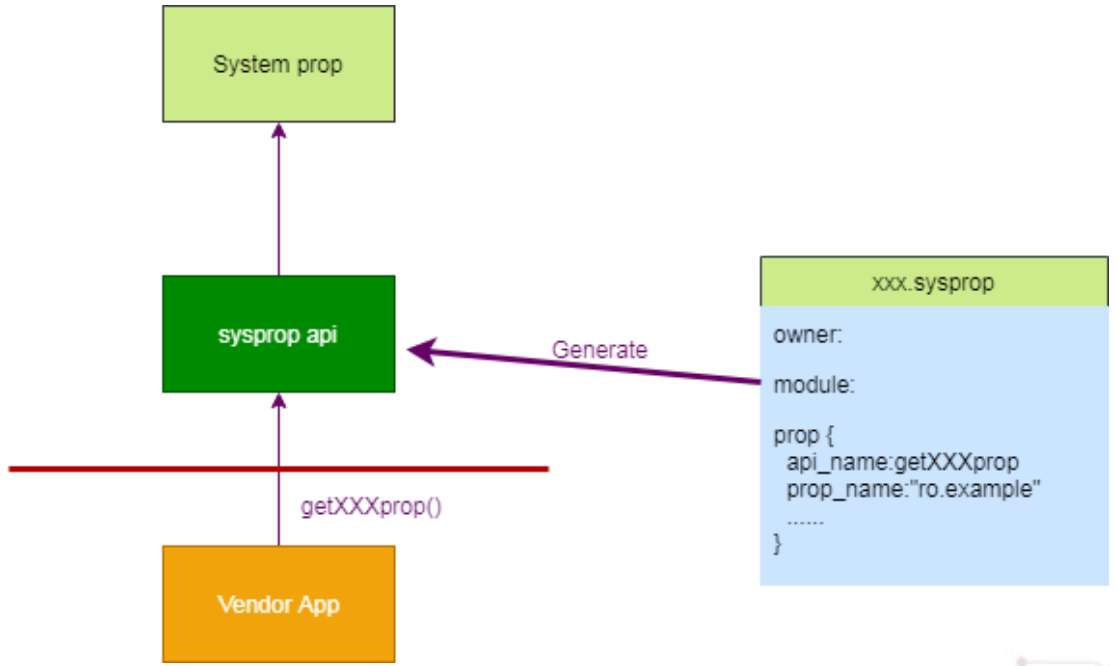
Google defines the Sysprop Description file (.sysprop). According to google suggestion, every prop that needs to be accessed across partitions needs to be implemented in this way.

Currently, this new feature on Android 10.0 does not enforce module to implement it.

Architecure

Sysprop as api architecture

- Explicitly defines properties
  - Concrete and Typed API across partitions
    - methods for Java / functions for C++
- Supports API Stabilization
  - Build will break if changed
- Gives consistency between C++ and Java world



New access method(1/2)

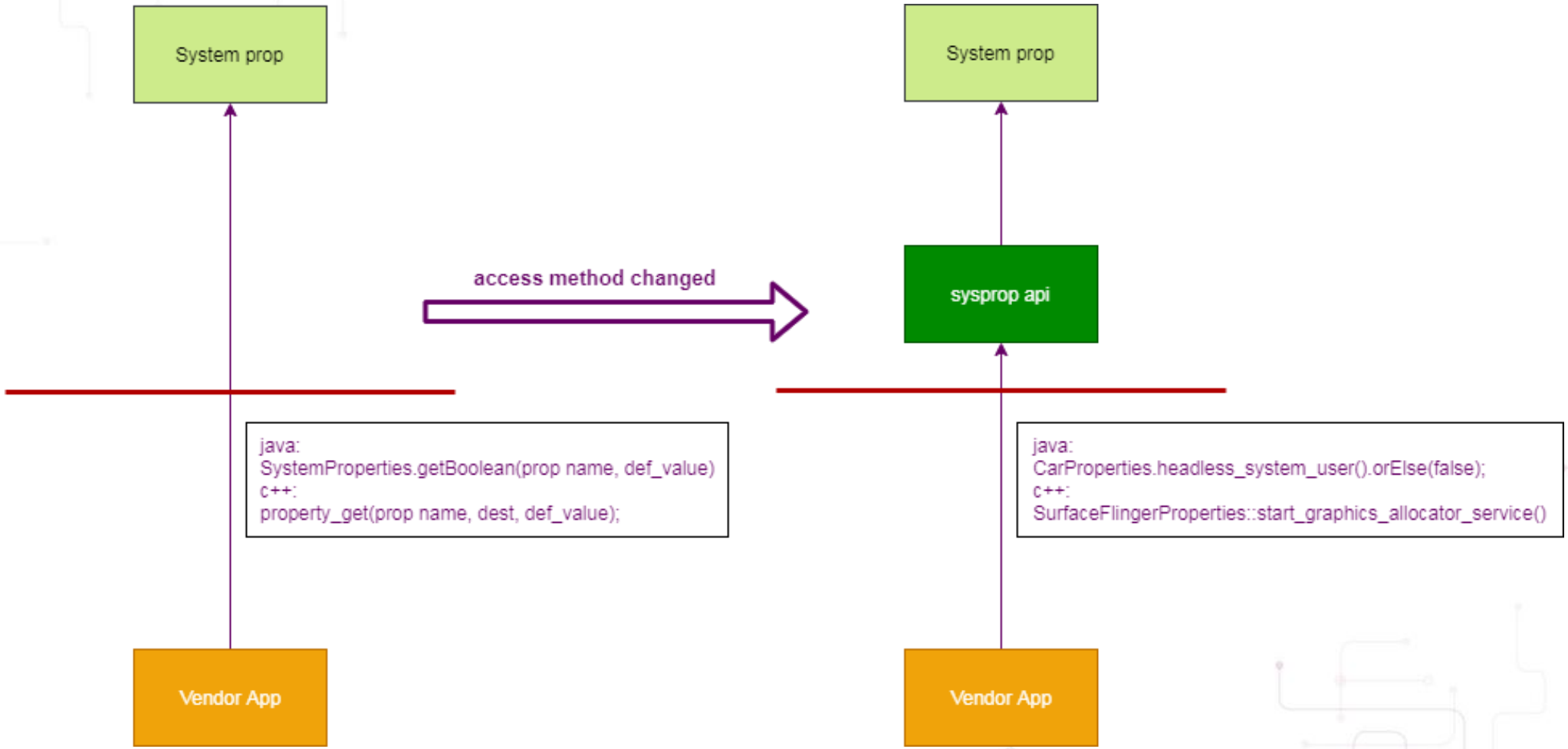
- Java

```
import android.sysprop.CarProperties;
CarProperties.headless_system_user().orElse(false);
```

- C++

```
#include <sysprop/SurfaceFlingerProperties.sysprop.h>
SurfaceFlingerProperties::start_graphics_allocator_service()
```

New access method(2/2)



Access restrictions

- Follow sysprop owner restrictions
  - Platform : Every partition can access platform-defined properties
  - Vendor : System can't access vendor's properties
  - Odm : Only vendor can access Odm-defined properties

  For details : frameworks/native/services/surfaceflinger/sysprop/api/system-current.txt

- Follow sysprop Scope restrictions
  - Internal: Only the owner can access.
  - System: Only bundled modules (not built against SDK or NDK) can access.
  - Public: Everyone can access, except for NDK modules.

- Follow Selinux restrictions

Write a .sysprop file(1/2)

The sysprop file mainly contains two parts.

- Properties
- Property

For the content of these two parts, Google has given the corresponding filling restrictions.

For details on the .sysprop file,
see system/tools/sysprop/sysprop.proto

```
message Properties {
Owner owner = 1;  //  ( Platform/Vendor/Odm )
module  string module = 2; // module name
repeated Property prop = 3; // Property
}
```

```
message Property {
  string api_name = 1;          // api name
  Type type = 2;                 // The type of the prop value
  Access access = 3;           // Read and write permissions
  Scope scope = 4;             // the  scope of api can use
  string prop_name = 5;       // prop name
  string enum_values = 6;    // enum values
  bool integer_as_bool = 7; //Allow integers to be used as bool
}
```

Write a .sysprop file(2/2)

An example of a Sysprop Description file：

```
owner: Platform
module: "android.sysprop.PlatformProperties"

prop {
    api_name: "date_utc"
    type: Integer
    prop_name: “persist.build.date_utc"
    scope: Internal
    access: ReadWrite
 }

prop {
    api_name: "build_date"
    type: String
    prop_name: "ro.build.date"
    scope: System
    access: Readonly
 }
```

Define sysprop_library modules with Sysprop Description files

```
sysprop_library {
    name: "PlatformProperties",
    srcs: ["android/sysprop/PlatformProperties.sysprop"],
    property_owner: "Platform",
    api_packages: ["android.sysprop"],
    vendor_available: true
}
```

How to use it(1/2)

Just like importing other so, we need to reference this defined sysprop in bp

```
java_library {
    name: "JavaClient",
    srcs: ["foo/bar.java"],
    libs: ["PlatformProperties"],
}
cc_binary {
    name: "cc_client",
    srcs: ["baz.cpp"],
    shared_libs: ["PlatformProperties"],
}
```

How to use it(2/2)

In the above example, you could access defined properties as follows.

Java example :

```
import android.sysprop.PlatformProperties;
static void foo() {
        Integer dateUtc = PlatformProperties.date_utc().orElse(-1);
}
```

C++ example :

```
#include <android/sysprop/PlatformProperties.sysprop.h>
using namespace android::sysprop;
void bar() {
        PlatformProperties::build_date(“2019-10-31”);
        std::string build_date = PlatformProperties::build_date().value_or("(unknown)");
}
```

Currently implemented

Sysprop as api related code path

- syste/tools/sysprop/ → parse the sysprop file to generate the corresponding api
- system/libsysprop/ → the storage path of the Platform-properties sysprop file
- build/soong/sysprop → build rule

THANKS

WWW.UNISOC.COM
紫 光 展 锐 科 技