Component Design

Android Deep Sleep(Suspend To RAM)

Harman International (India) Pvt. Ltd.

**Contents**

[1. Introduction 3](#_Toc495051102)

[1.1 Purpose 3](#_Toc495051103)

[1.2 Scope 3](#_Toc495051104)

[1.3 Definitions, Acronyms and Abbreviations 3](#_Toc495051105)

[1.4 References 3](#_Toc495051106)

[1.5 Overview 3](#_Toc495051107)

[1.5.1 Differences to Mobile phone use case 4](#_Toc495051108)

[1.6 Design Approach 4](#_Toc495051109)

[2. Architectural Representation 7](#_Toc495051110)

[3. Components and Frameworks Reused 8](#_Toc495051111)

[4. Component Design 9](#_Toc495051112)

[4.1 CarService 9](#_Toc495051113)

[Module Prologue 9](#_Toc495051114)

[Name and Description 9](#_Toc495051115)

[Function 9](#_Toc495051116)

[Interfaces (relationships with other modules / components) 9](#_Toc495051117)

[Physical data structure/data file descriptions 9](#_Toc495051118)

[Process (pseudo-code algorithm, PDL) 9](#_Toc495051119)

[4.2 Vehicle HAL 9](#_Toc495051120)

[Module Prologue 9](#_Toc495051121)

[Name and Description 9](#_Toc495051122)

[Function 9](#_Toc495051123)

[Interfaces 9](#_Toc495051124)

[Physical data structure/data file descriptions 10](#_Toc495051125)

[Process (pseudo-code algorithm, PDL) 10](#_Toc495051126)

[4.3 Vehicle Power Moding Service(VPS) 10](#_Toc495051127)

[Module Prologue 10](#_Toc495051128)

[Name and Description 10](#_Toc495051129)

[Function 10](#_Toc495051130)

[Interfaces 10](#_Toc495051131)

[Physical data structure/data file descriptions 11](#_Toc495051132)

[Process (pseudo-code algorithm, PDL) 11](#_Toc495051133)

[5. Formulas and Algorithms 11](#_Toc495051134)

[6. Diagrams Representations 12](#_Toc495051135)

[7. Resources 13](#_Toc495051136)

[8. Requirement Traceability Matrix 13](#_Toc495051137)

[9. Change History 14](#_Toc495051138)

# Introduction

## Purpose

This document explains the implementation of Android Deep Sleep Suspend to RAM (S2R) and the possible approaches of S2R notifications to apps/services or other framework components.

## Scope

This document explains the Android deep sleep (S2R) mechanism and approaches for S2R notifications to apps/services.

## Definitions, Acronyms and Abbreviations

| **Acronym** | **Expansion** |
| --- | --- |
| S2R | Suspend To RAM |
| AOSP | Android Open Source Project |
| IOC | I/O Controller |
| SOC | System On Chip |
| IPC | Inter Process Communication |
| VPS | Vehicle Power Moding Service |
| VHAL | Vehicle Hardware Abstract Layer |
| CAN | Controller Area Network |
| LIN | Local Interconnect Network |
| MOST | Media Oriented Systems Transport |
| API | Advanced Power Management |
| ACPI | Advanced Configuration and Power Interface |
| SPM | System Power Management |

## References

| **#** | **References** | **Notes** | **Version** |
| --- | --- | --- | --- |
| 1. | TRS Titan Suspend to RAM (S2R) |  | A.04 |
| 2. |  |  |  |
| 3. | https://source.android.com/devices/automotive/ |  |  |

## Overview

Power management in operating systems is important due to the ever increasing power demand of mobile devices especially cell phones & embedded devices. In order to reduce wasted power, multiple hardware power saving features are employed by Linux such as clock gating, voltage scaling, activating sleep modes and disabling memory cache. Each of these features reduces the system’s power consumption at the expense of latency and/or performance.

These tradeoffs on a Linux system are managed by either Advanced Power Man+agement (API) or Advanced Configuration and Power Interface (ACPI).

Linux also has a functionality called System Power Management (SPM): the process of placing the entire system into a low power state. In a low power state, the system is consuming a small, but minimal amount of power yet maintaining a relatively low response latency to the user. The exact amount of power and response latency depends on the state the system is in.

The states a system can enter on Linux are dependent on the underlying platform, and differ across architectures and even generations of the same architecture. There tend to be three states that are found on most architectures that support a form of SPM, though. The kernel explicitly supports these states -Standby, Suspend, and Hibernate, and provides a mechanism for a platform driver (an architectural port of the kernel) to define new states.

1. **Standby**

Standby is a low-latency power state that is sometimes referred to as “Power-on suspend”. In this state, the system conserves power by placing the CPU in a halt state and the devices in the state. The power savings are not significant, but the response latency is minimal typically less than 1 second.

1. **Suspend**

Suspend is also commonly known as” Suspend-to-RAM”. In this state, all devices are placed in sleep state except main memory, is expected to maintain power. Memory is placed in self-refresh mode, so its contents are not lost. Response latency is higher than Standby, yet still very low between 3-5 seconds.

1. **Hibernate**

Hibernate conserves the most power by turning off the entire system, after saving state to a persistent medium, usually a disk. All devices are powered off unconditionally

Suspend to RAM is brought in to be able to almost instantly provide a fully functional infotainment system compared to a cold start which takes longer time to make system fully functional.

Just like how you don't have to wait to make a call after you bring a phone out of standby.

Suspend to RAM feature is supported by the Linux kernel. The goal of this feature is to be able to freeze all applications and thaw them at a later stage.

**1.5.1 Differences to Mobile phone use case**

The suspend use-case of a mobile phone is considerably different from the use-cases in the automotive environment. In the case of a mobile phone, the system goes into a low power mode by turning off subsystems and clocking down the CPU.  The CPU continues to run in a low power mode. This brings down the power usage considerably, extending the standby time of the phone.   However, standby currents required in the car as much lower than what is allowed on mobile phones. Even though the battery in the car is much larger than that on the mobile phone, the head-unit is not the only ECU that needs to be in standby.  Current budgets are allocated and distributed across multiple ECUs.

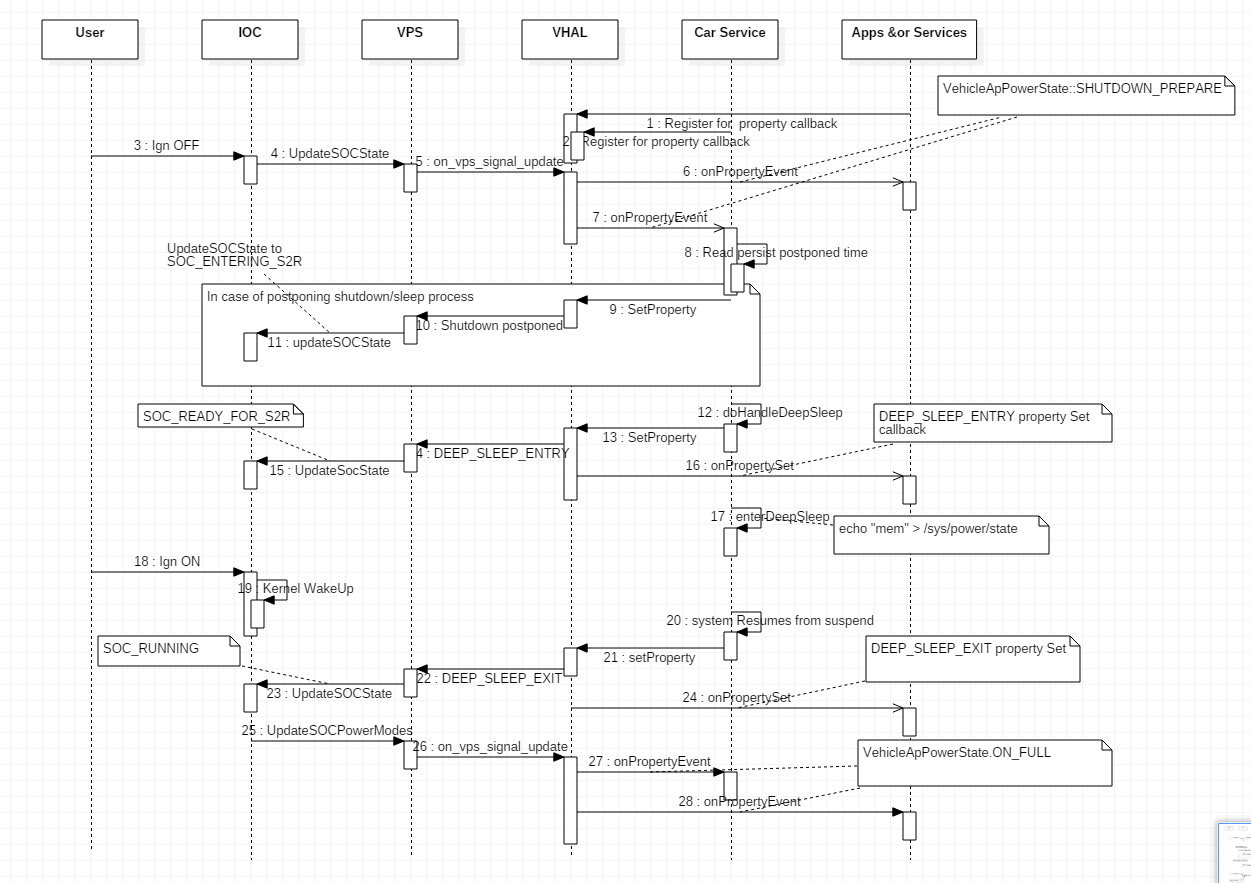
Due to this limitation, Suspend to RAM approach is taken. This is similar to the case when you put your PC into suspend. The contents of the memory are retained and the CPU is shutdown. Peripherals are brought to a low power state or shutdown if a low power state is not possible.

## Design Approach

As of now there are three ways of S2R notifications to application/services

1. Through vehicle HAL properties callback.
2. Exposing binderized interface of CarPowerManagerService to Application/Services.
3. Broadcasting intents.

**Approach 1# Through Vehicle Hal properties** **callback. (Preferred approach)**



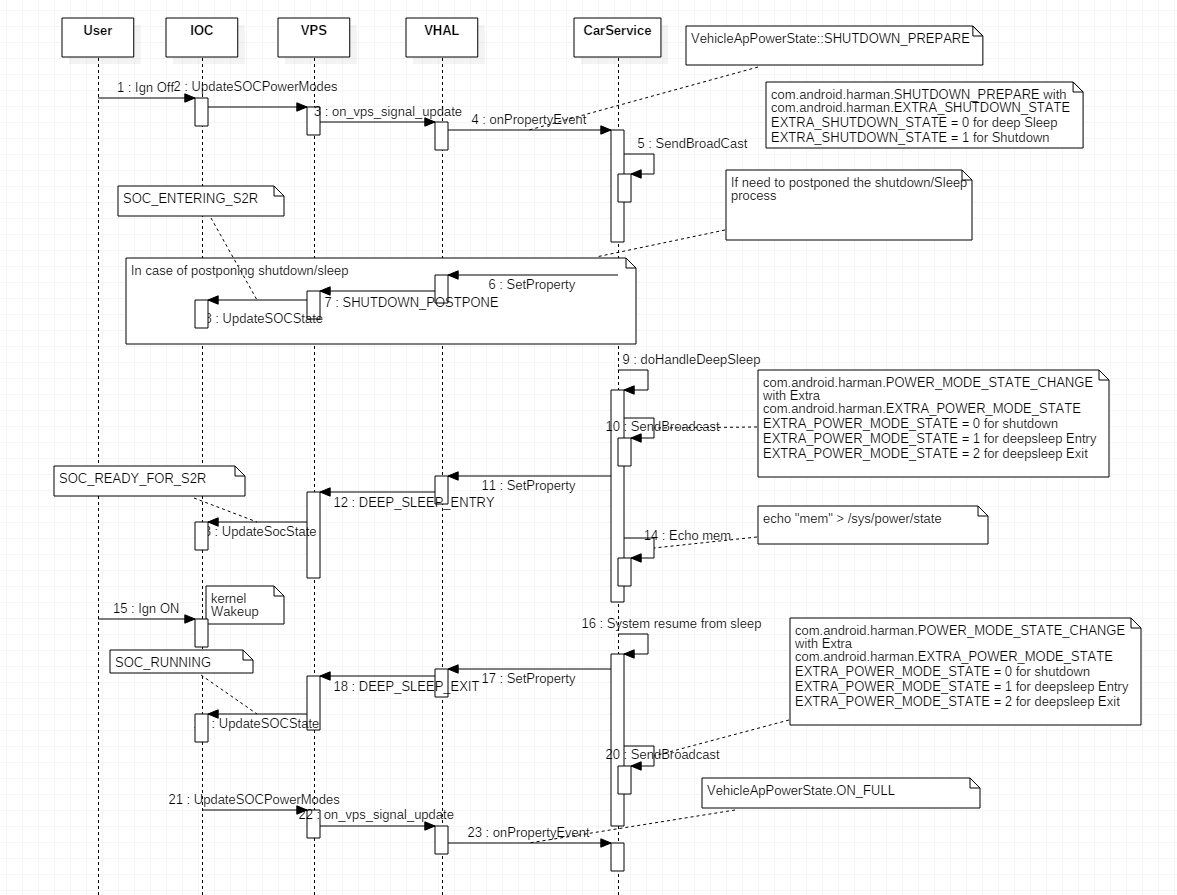
* System app’s/services register to vehicle HAL for AP\_POWER\_STATE property
* CarService also register for vehicle HAL for properties as a initialization process
* **S2R Entry**
  1. IOC sends S2R Request to VPS.
  2. VPS will update the VHAL for **VehicleApPowerState::SHUTDOWN\_PREPARE**
  3. VHAL updates the CarService/Apps/services for **VehicleApPowerState::SHUTDOWN\_PREPARE** property.
  4. Based on this above notification, Apps/Services prepares for Shutdown/Sleep.
  5. **Car Service reads a persist property for shutdown/sleep postpone time (Need to discuss this for shutdown postpone)**
  6. If need to postpone shutdown/sleep, then CarService inform the VHAL about VehicleApPowerSetState::SHUTDOWN\_POSTPONE which informs the VPS for shutdown postpone.
  7. After Postpone time, inform VPS to enter to deep sleep state through VHAL property DEEP\_SLEEP\_ENTRY. VPS in turn update SOC state to IOC saying, SOC is ready for S2R.
  8. VHAL informs the DEEP\_SLEEP\_ENTRY property Set to apps/services which had registered for set event.
  9. Car Service then makes SOC to suspend state by writing “mem” to /sys/power/state.
* **S2R Exit**
  1. On ignition, IOC wakes up the System (kernel).
  2. CarService resumes from sleep on kernel wakeup & inform VPS about deep sleep exit through VHAL property DEEP\_SLEEP\_EXIT. VPS in turn update the SOC state to IOC saying SOC is Running.
  3. VHAL informs the DEEP\_SLEEP\_EXIT property set to app/services which had register for set event
  4. VHAL sends property(VehicleApPowerState::ON\_FULL) change to CarService
  5. CarService wakes up the AOSP PowerManager

**Approach 2# Exposing Bindarized interface of CarPowerManagementService**

With this approach, app’s/services instead of registering to vehicle HAL for property event they have to register with CarPowerManagementService for PowerServiceEventListener & PowerEventProcessingHandler.

**This approach needs lot of code changes in AOSP car package in order to expose binderized interface of CarPowerManagementService**

**Approach 3# Broadcasting Intents for S2R events from CarService**



# Architectural Representation

IOC

VPS

Linux Android (SOC side)

Kernel space

Android User space

Car Service

Sysfs

Kernel PM

Device drivers

Hardware (SOC & Peripherals )

Suspend

command

Resume signal

Android Apps/Services

Vehicle HAL

Suspend Path

Resume Path

# Components and Frameworks Reused

|  |  |  |  |
| --- | --- | --- | --- |
| Reused Component or Framework | Supplier | Benefits List | Repository Source |
| Car Service | AOSP |  | /packages/services/Car |
| VPS | HARMAN |  | /vendor/Harman/titan/packages/power-moding |
| Harman Vehicle HAL | HARMAN |  | /vendor/harman/titan/hardware/automotive/vehicle/v1\_0 |

# Component Design

## CarService

### Module Prologue

CarService is AOSP framework component which is responsible for car related functionalities.

### Name and Description

CarService is the client to vehicle HAL, and it has list of HAL services from HWservicemanager via hardware binder.

CarService and VHAL interact via HIDL interface.

### Function

Handles applications request related to CAR functionalities & reponds to any vehicle property changes.

### Interfaces (relationships with other modules / components)

### Physical data structure/data file descriptions

* External data dependencies
* Internal data descriptors

### Process (pseudo-code algorithm, PDL)

## Vehicle HAL

### Module Prologue

Vehicle HAL provides a consistent interface to the Android framework regardless of physical transport layer Like CAN, LIN or MOST.

### Name and Description

Vehicle HAL module enable applications to subscribe, unsubscribe, set, get & update Google supported vehicle properties and Harman Custom properties

### Function

Monitors the CAN BUS & updates the CarService and other Apps/Services for any property change and provides a mechnism to set the vehicle property.

### Interfaces

1. IVehicle interface

|  |  |  |
| --- | --- | --- |
| Return Type | Name | Description |
| vec<VehiclePropConfig> | getAllPropConfigs | Returns a list of all property configurations supported by this vehicle HAL |
| vec<VehiclePropConfig> | getPropConfigs | Returns a list of property configurations for given properties. |
| VehiclePropValue | get | Get a vehicle property value |
| StatusCode | set | Set a vehicle property value |
| StatusCode | subscribe | Subscribes to property events |
| StatusCode | unsubscribe | Unsubscribes from property events |

1. IvehicleCallback

|  |  |  |
| --- | --- | --- |
| Return Type | Name | Description |
| void | onPropertyEvent | Event callback happens whenever a variable that the API user has subscribed to needs to be reported |
| void | onPropertySet | This method gets called if the client was subscribed to a property using SubscribeFlags::SET\_CALL flag and IVehicle#set(...) method was called |
| void | onPropertySetError | Set property value is usually asynchronous operation. Thus even if client received StatusCode::OK from the IVehicle::set(...) this doesn't guarantee that the value was successfully propagated to the vehicle network. If such rare event occurs this method must be called |

### Physical data structure/data file descriptions

* External data dependencies
* Internal data descriptors

### Process (pseudo-code algorithm, PDL)

## Vehicle Power Moding Service(VPS)

### Module Prologue

VPS maintains the vehicle power moding states

### Name and Description

Vehicle power moding service is SOC components which is power moding interface between VHAL & IOC

### Function

Vehicle power moding service is responsible for interacting with IOC & handling shutdown request from IOC and as well as sending SOC’s heartbeat to IOC. VPS also handle CarService/TOD request like setting RTC to IO

### Interfaces

VPS 🡨🡪 IOC

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| ssize\_t | ipc\_read | VPS reads the following messages from IOC  Shutdown request message  Boot reason message  Receive RTC time message |
| ssize\_t | ipc\_write | VPS writes the following messages to IOC  Request RTC time  Set RTC and Wakeup time  Boot Complete |

VPS Server 🡨🡪 Vehicle HAL(VPM Client)

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| Class | BpInterface<IHarmanPowerManagement> | Transact VPM SET property setVehiclePowerModeProperty |
| Class | BpInterface<IHarmanPowerManagement> | Transact VPM GET property getVehiclePowerModeProperty |

VPS Client 🡨🡪 Vehicle HAL(VPM Server)

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| Void | onTransact🡪OnchangePowerModeProperty | Updates the power mode change to VHAL which registers for on\_vps\_signal\_update |
| Void | setVehiclePowerModeProperty | Transact the request over IharmanPowerManagement interface |
| Int\_32 | getVehiclePowerModeProperty | Get the power mode property. |
| void | on\_vps\_signal\_update | Update the power moding state to car service |

### Physical data structure/data file descriptions

* External data dependencies
* Internal data descriptors

### Process (pseudo-code algorithm, PDL)

# Formulas and Algorithms

* Formulas and Algorithms Not Documented in the any other documents

# Diagrams Representations

Approach 1: Notifying Apps/Service through Vehicle Hal properties

Apps/Services

Vehicle Hal

VPS

IOC

Car Service

Android FW

Kernel PM

Apps/Services

Vehicle Hal

VPS

IOC

Car Service

Android FW

Kernel PM

Approach 2: Notifying Apps/Service through Binderized Car Service Interface

# Resources

Memory (RAM) usage

Storage Requirement

CPU Load

### 

# Requirement Traceability Matrix

***<<Please refer the Project Requirements Traceability Matrix >>***

# Change History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Description | Created by / Date  (DD-MMM-YY) | Reviewed & Approved By / Date (DD-MMM-YY) |
| 1.0 | Initial Version | Riyaz Ahmed  (06-OCT-17) |  |
|  |  |  |  |