Version 1.0
Production Readiness Specification Requirements Definition

Sep 30, 2021

Contents

[Change history]

[No]	[Date]	[Version]	[Description]	[Changer]
1	**2021/9/30**	**v1.0**	**First edition**	**Riku Nomoto(Woven Alpha, Inc.)**

1. Purpose

Automotive Grade Linux(AGL) is a Linux Foundation Workgroup dedicated to create open source software solutions for automotive. In the activity of AGL, by targeting In-Vehicle-Infotainment(IVI) systems, Production Readiness activity has started from October in 2020. Production Readiness is an activity to ensure that AGL's software meets the requirements and quality as required by the IVI in-vehicle.

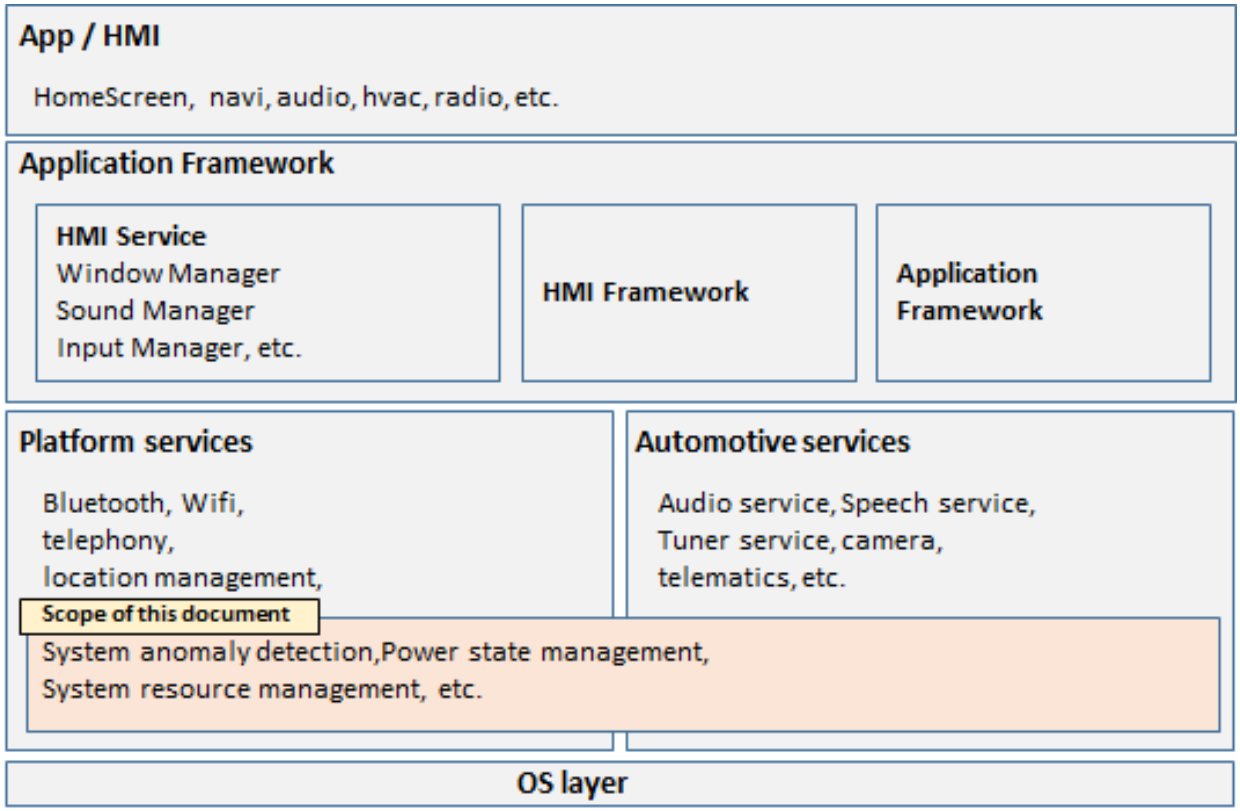
As a part of this activity, we conduct an activity to describe the function requirements that would be commonly required for each Automotive company's IVI products.

2. Scope

AGL published a requirement specification as AGL Requirement Specification v1.0. However, it has been six years since it was published and there is a gap between the source code and the specification document. Also, “Requirements for Production” is not clearly defined. In order to resolve these issues, IVI-EG have created ProductionReadinessProfile to disclose the source code of actual products based on the principle of "Code first", and developed the requirements specification that corresponds with the code.

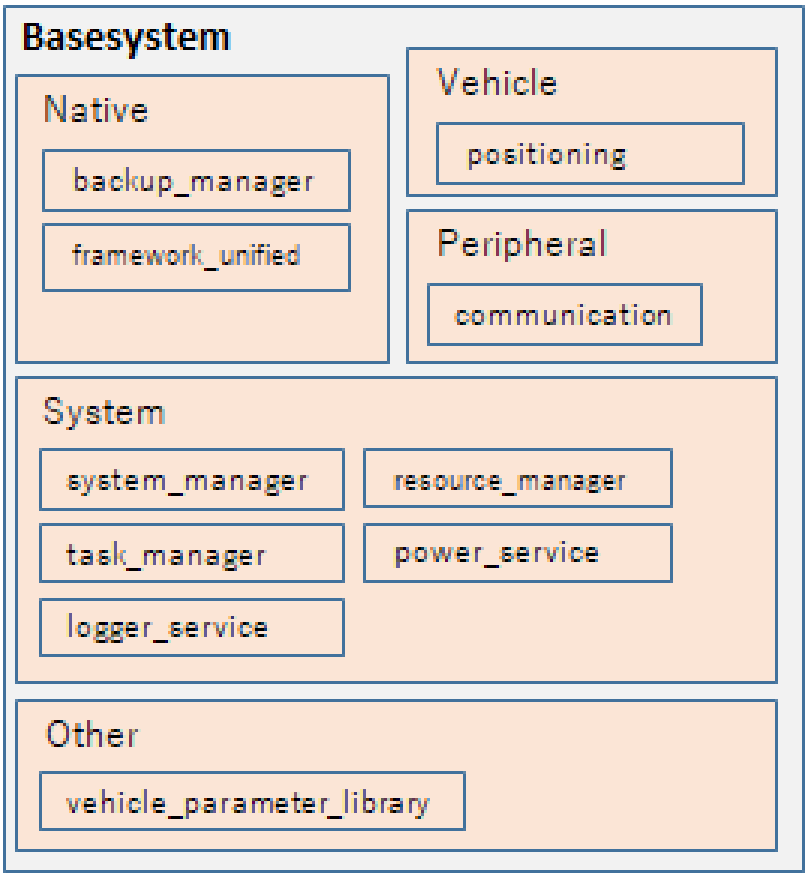
The figure 1 is a diagram created based on the architectural diagram described in that document, and this document targets the service layer.

Figure1) Scope of this document



This is the software architecture diagram of Basesystem that is platform services based on actual IVI products, and that is open sourced through IVI-EG activity. This document refers to Basesystem as examples of the implementation.

Figure 2) Basesystem unit



3. Glossary

Here is an explanation of terminology.

Table3

#	Item	Description
1	Basesystem	A group of software located in a layer close to Operating System. It has been contributed to AGL in March 2021.

2	IVI	In-Vehicle-Infotainment. On-board equipment which has navigation, audio functions, and so on.
3	IVI system	The system for booting IVI up.
4	ACC(Accessory)	The power source that supplies electricity to the IVI and other electrical components.When ACC is turned on, the information is sent to IVI and IVI is activated. In addition to ACC, there are other power sources such as +B and IG in general.
5	HMI	Human Machine Interface. In particular, this document refers to the Interface, which is operated by the user on the IVI.
6	Heartbeat	The function to confirm the behaviors by the regular communication between the two processes.
7	Resident service	A service that launches automatically at system startup. The Resident service is always running while the IVI is running, and provides functions as needed.
8	Non-resident service	A service that is activated by a request from an application. The non-resident service is not activated except when it is requested to be activated as needed.
9	Service launcher	A function to launch services.
10	Service terminator	A function to terminate services.
11	Service Failure Detection and Procedure module	The module with the ability to detect failure in system and provide the procedure like restoring or storing logs.
12	System Logging Support module	The module to leave a log in a specified location in the IVI system when a request to leave a log is received.

13	Backup Data Management module	The module with the ability to store data permanently in the IVI system.
14	Vehicle Parameter Configuration module	The module with the ability to provide necessary information in response to a request from a service to acquire necessary environmental variables, or to determine whether an IVI product supports various functions and return the results to the service.
15	CAN communication module	The module for CAN data processing between other ECUs and applications.
16	GPS	Global(Satellite) Positioning System
17	Sensor data	Gyroscope, Accelerometer, Speed pulse, vehicle reverse info, etc.
18	GPS and Sensor Communication module	The module with the ability to get and send GPS data or Sensor data to necessary applications.

4. Use cases and Functional requirements

4.1. Power State Management

4.1.1. Abstraction

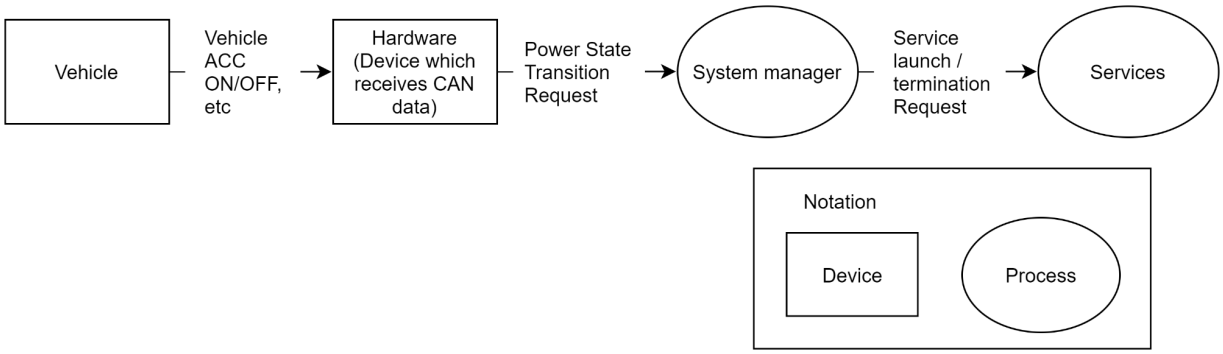
A requirement of IVI systems is that they need to support transitions to multiple power states. Not just a simple power off/on state, but several other types of power states are required. In each power state, it is necessary to switch the function to be controlled and the power state of connected devices.

Due to the diversity of use cases in modern automotive, it is necessary for IVI systems to operate even in situations where there is no voltage supplied by the battery. A problem inherent to IVI is the limited power supplied by the battery. If you always start and maintain any hardware in the state, it will lead to battery failure. Therefore, when using IVI, it is necessary to define the various power states according to the use case of each user, and in each power state, activate the minimum amount of hardware necessary to realize the use case and realize the function.

The IVI receives power state transition requests from the vehicle (hardware) side, notifies these requests to the various services, and performs state transitions. In this 4.4 section, the use cases involving power state transitions, the functional requirements for realizing those use cases, and the functions of the Basesystem as a sample implementation are described.

The description of Hardware in 4.4 section indicates some kind of hardware (such as CAN signal) that notifies the IVI side of power state transition.

Figure 4



4.1.2. Use cases

In the Table 5, use cases which need the Power State Management function are described. In each use case, the transition of the car's power state is changed, and the IVI should be changed its state accordingly.

Table 5

#	Item	Description
UC.PS.1	IVI(quick) startup	The driver opens the door and gets into the car. And the user presses the button for ACC-ON and the car and the IVI is turned on.
UC.PS.2	IVI shutdown	The driver presses the button for ACC-OFF, gets out of the car and does not get into the car for a long time.
UC.PS.3	Startup unlinked with the ACC	The driver presses a button for IVI-ON and uses audio, video, and communication services without ACC-ON.
UC.PS.4	Delayed ending	After arrival at the destination, when the driver wants to continue a handsfree call after ACC-OFF, the user continues it although the display is off.
UC.PS.5	Remote parking system	When the driver wants to get the car out of the parking lot, the driver uses the smartphone-linked function to control outside and gets into the car.
UC.PS.6	Demotion assistance	After the driver arrives at the destination, the driver presses the button for ACC-OFF and gets out of the car. When the driver gets out of the car, if there is danger behind the car, an alarm sounds and the driver is notified.
UC.PS.7	Return to the car soon	The driver presses the button for ACC-OFF, gets out of the car and gets into the car soon.
UC.PS.8	Return to the car temporally	The driver opens the door and takes out the luggage. The driver closes the door and does not get into the car.
UC.PS.9	Automatic updates via OTA	IVI automatically updates software through OTA function while the driver is away from the car.

UC.PS.10	Data backup	IVI performs data backup while the driver is away from the car.

In Production Readiness, the power state transition of IVI in general is defined as follows.

1. The state of IVI power off(Power-off)
2. The state which is ready to boot IVI(Ready)
3. The state which the part of IVI features can be used(Partially running)
4. The state which IVI has started(Running)

Although the four states are considered necessary for product use cases, the 2 and 3 states depend on the requirements of each OEM.

The following table is a description of the above four states.

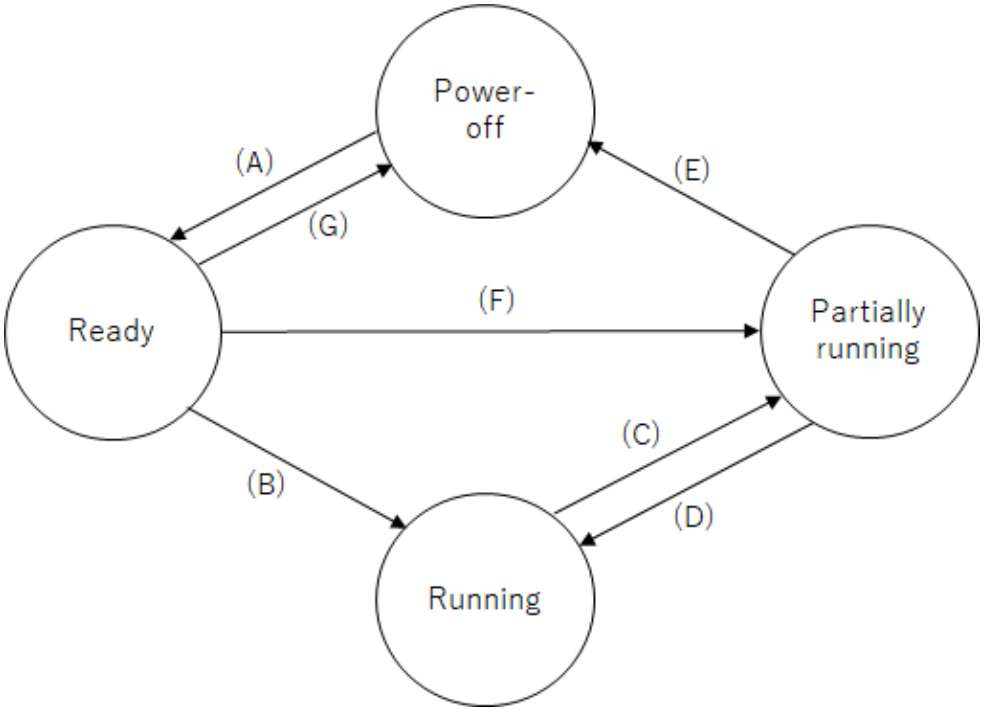
Table 6

#	IVI power state	Description of each state
ST.PS.1	Power-off	This is the state in which the IVI system is turned off and the IVI display is turned off.
ST.PS.2	Ready	This is the state in which the IVI system is turned on and the IVI display is turned off. The IVI system prepares selected services so that the IVI system can provide user functions quickly when the IVI is turned on. ※When starting a car with IVI unprepared, it takes a lot of time. Therefore, the “Ready” status is necessary.
ST.PS.3	Partially running	This is the state in which the IVI system is turned on. When transitioning to this state from any other state, the IVI system will be able to use the selected services for a while. (There is

		no specific mention of turning the IVI display off/on. It shall be dependent on implementation.)
ST.PS.4	Running	This is the state in which the IVI system is turned on and the IVI display is turned on. The ACC is on and all of the functions can be used.

The following shows the IVI state transition diagram of the above table. The conditions for each transition are also described.

Figure 7) State transition diagram



When the power status of the vehicle changes, the power status of the IVI will transition. The cause of the change in the vehicle power state depends on the vehicle signal, CAN signal, vehicle system, etc. Therefore, the required state/transition may be different depending on the product and hardware requirements of each OEM. However, the above state transition diagram shows the state that realizes the use cases defined in this document. Each OEM should add or delete state/transition as necessary.

Table 8

Use case	State transition diagram	The condition of transition
(1)IVI (quick) startup	(A), (B)	(A)The transition request to change the selected services states to “Ready” is sent from the Hardware side. (B)The transition request to change the IVI state to “Running” is sent from the Hardware side.
(2)IVI shutdown	(C), (E)	(C)The transition request to change the IVI state to “Power-off” is sent from the Hardware side. (E)When no transition request is received and a certain amount of time has passed, the state is changed.
(3)Startup unlinked with the ACC	(A), (F)	(A)The transition request to change the selected services states to “Ready” is sent from the Hardware side. (F)The transition request to change the selected services states to “Partially Running” is sent from pressing a button for IVI-ON.
(4)Delayed ending	(C)	(C)When the transition request to change the IVI state to “Power-off” is sent from the Hardware side, the state goes through “Partially running” once. If there is a request to execute the function, it is done in this state.
(5)Remote parking system	(A), (F)	(A)The transition request to change the selected services states to “Ready” is sent from the Hardware side. (F)The transition request to change the selected services states to “Partially Running” is sent from the user’s smartphone.
(6)Demotion assistance	(C)	(C)When the transition request to change the IVI state to “Power-off” is sent from the Hardware side, the state goes through “Partially running” once. If there is a request to execute the function, it is done in this state.
(7)Return to the car soon	(C), (D)	(C)The transition request to change the IVI state to “Power-off” is sent from the Hardware side.

		(D)The transition request to change the IVI state to Running is sent from the Hardware side.
(8)Return to the car temporally	(A), (G)	(A)The transition request to change the selected services states to “Ready” is sent from the Hardware side. (G)When no transition request is received and a certain amount of time has passed, the state is changed.
(9)Automatic updates via OTA	(A), (F), (E)	(A)The transition request to change the selected services states to “Ready” is sent from the Hardware side. (F)The transition request to change the selected services states to “Partially Running” is sent from pressing a button for IVI-ON. (E)When no transition request is received and a certain amount of time has passed, the state is changed.
(10)Data backup	(A), (F), (E)	(A)The transition request to change the selected services states to “Ready” is sent from the Hardware side. (F)The transition request to change the selected services states to “Partially Running” is sent from pressing a button for IVI-ON. (E)When no transition request is received and a certain amount of time has passed, the state is changed.

4.1.3. Functional Requirements

The Table 9 includes the functional requirements of Power State Management module.

Table 9

#	Item	Related use case	Description

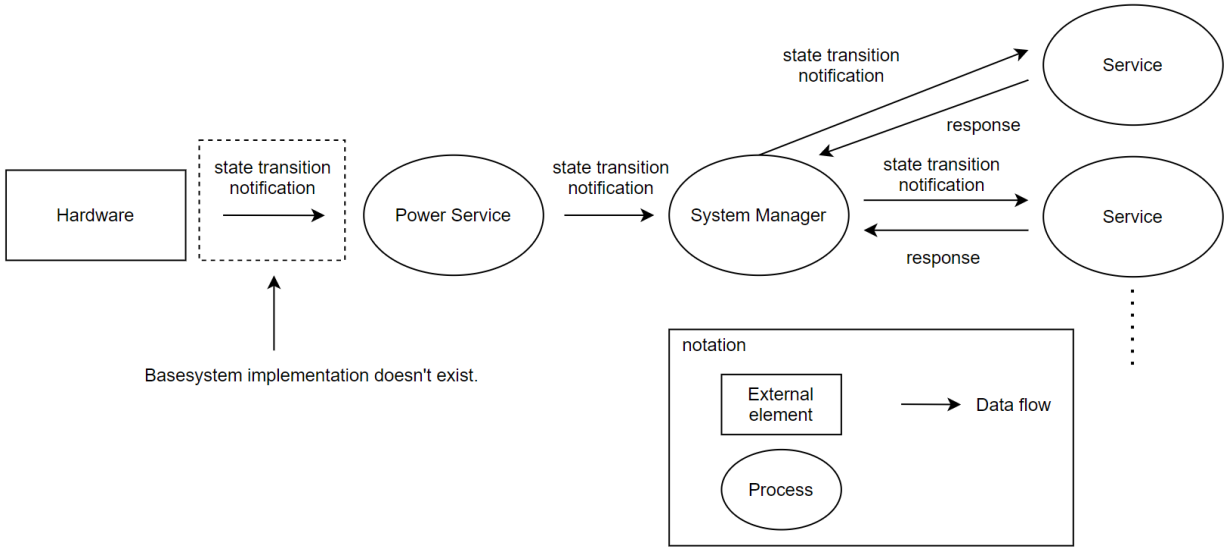
FR.PS.1	Power state receiving	All of the use cases	Power State Management should receive state transition requests from the Hardware side.
FR.PS.2			The incoming trigger of the power state depends on each OEM so the incoming trigger should be flexible enough to change.
FR.PS.3	Power state sending		Power State Management should notify the power state change request to each service.
FR.PS.4			Since the number(definition) of power states depends on each OEM, adding or removing power states should be flexible enough to change.
FR.PS.5			Only services which require control by power state change requests should be notified of the necessary request.
FR.PS.6	Device control		According to the power state transition, each application or middleware related to services should switch the function to activate or the control devices.

4.1.4. Power State Management in Basesystem

4.1.4.1. Reference implementation by Basesystem

In the implementation of Basesystem, the function modules for Power Service Management are Power Service and System Manager. The software configuration diagram is shown in Figure 2. As shown in Figure 10, Power service provides functions such as notifying System manager of power state transition requests by getting the notification from the hardware side. In order to control the system according to the power state transition request, System manager notifies the services of a power state transition received from Power service.

Figure 10



4.1.4.1.1. Power Service

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/system/power_service;h=fabcbf6aabbf0b7dd8a1df2fb35491029a2d1fdd;hb=refs/heads/master

4.1.4.1.2. System Manager

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/system/system_manager;hb=refs/heads/master

4.2. Service Launch and Termination

4.2.1. Abstraction

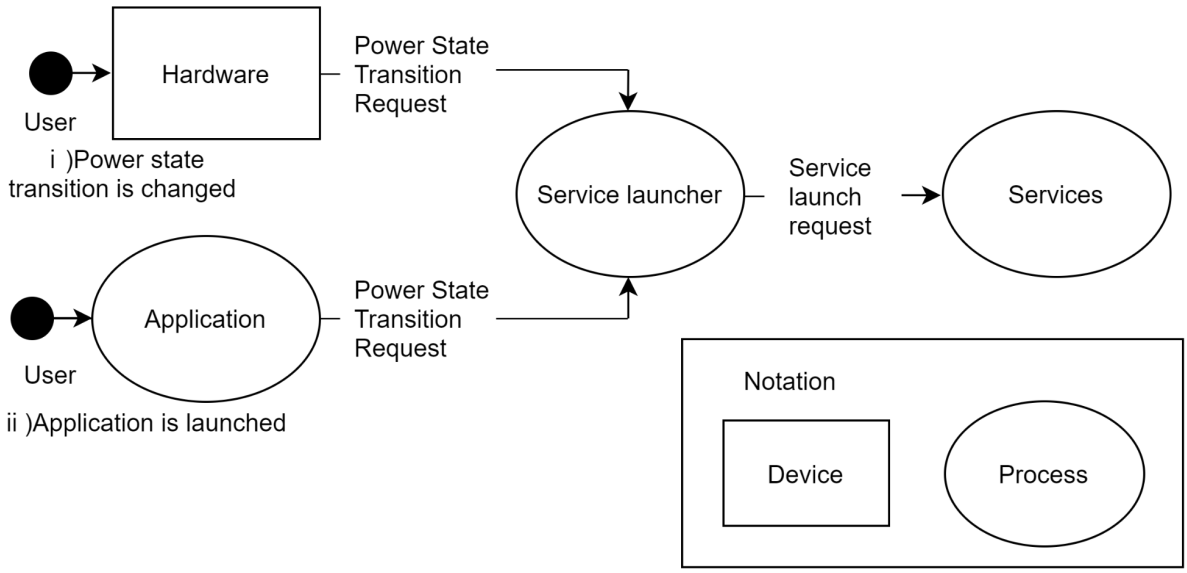
IVI realizes its functions through coordination of multiple services. Since resources such as CPU and memory are finite, it is necessary to launch and terminate specific services depending on the use case, rather than always launching and terminating all services.

There are two main triggers for launching services(Figure11): the first is when any IVI service is required to launch along with a power state transition(ⅰ), and the second is when the application is launched and the service is requested to be launched along with the application launch(ⅱ). When the service launcher receives information from each of these triggers, it will make a launch request to the required service.

Similarly, there are two main triggers for terminating services: the first is when any IVI service is required to terminate along with a power state transition, and the second is when the application terminates and the service is no longer needed (implementation depends on each company).

This chapter describes the use cases with launching and terminating services, the functional requirements to realize the use cases, and the current Basesystem design and implementation as a reference.

Figure 11) The trigger of launch service



4.2.2. Use cases

In the Table 12, use cases which need the System Launch and Termination module for services are described.

Table 12

#	Item	Description
UC.LT.1	Service launch with power state transition	There are multiple use cases accompanied by service launch. ● The driver presses the button for ACC-ON, the IVI starts and the home screen is launched (the services required to launch the home screen are launched when the IVI starts). ● IVI automatically updates software through OTA function while the driver is away from the vehicle. ● IVI backs up data while the driver is away from the vehicle, etc.

UC.LT.2	Service launch when using application	The driver uses the navigation application and sets the destination. The user starts the application, enters the destination, and the time required and route are displayed.
UC.LT.3	The launch order of previously used service	When the driver starts up the system, the driver uses the services that were running when the system was last stopped before any other services.
UC.LT.4	Service termination after finishing application	The driver makes a hands-free phone call while driving the car. After the user finishes the call, the service that is no longer needed is also terminated by being requested.
UC.LT.5	Service termination with power state transition	There are multiple use cases accompanied by service termination. ● The driver arrives at the destination and stops the car. The IVI system stops and the activated services are terminated. ● After arrival at the destination, when the driver wants to continue a handsfree call after ACC-OFF, the user continues it although the display is off. ● After the driver arrives at the destination, the driver presses the button for ACC-OFF and gets out of the car. When the driver gets out of the car, if there is danger behind the car, an alarm sounds and the driver is notified, etc.
UC.LT.6	The order of service termination	The driver arrives at the destination and stops the car. When the driver presses the button for ACC-OFF, the system will exit with a log of the user's system usage at the end.

4.2.3. Functional Requirements

The Table 13 includes the functional requirements of the Services Launch and Termination module.

Table 13

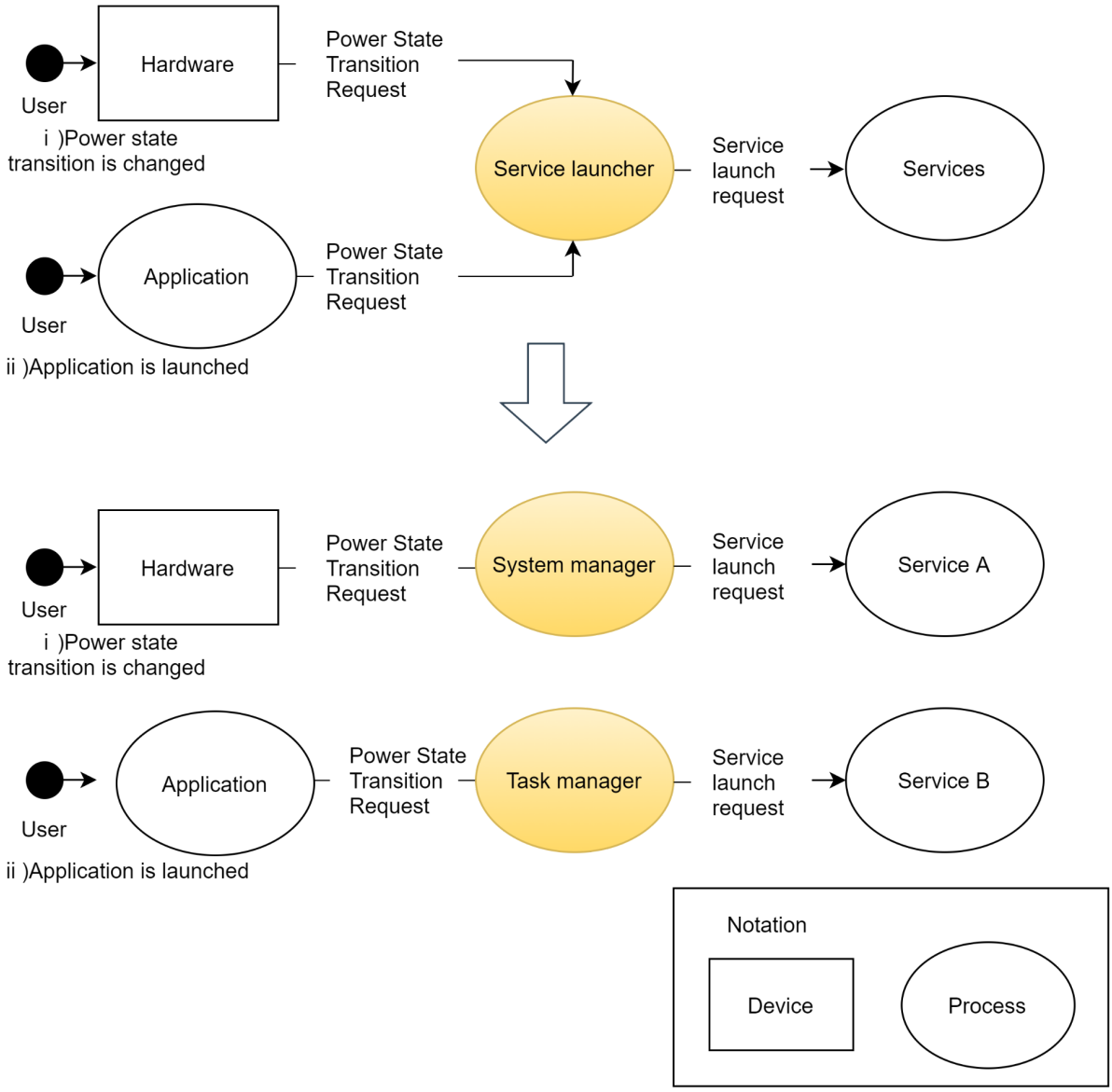
#	Item	Related use case	Description
RQ.LT.1	Service launch	UC.LT.1	Service launcher shall send launch requests to the required services according to the power state transition requests.
RQ.LT.2		UC.LT.1	Service launcher shall allow the configuration of which IVI services to launch.
RQ.LT.3		UC.LT.1	The order of the necessary services shall be configurable.
RQ.LT.4		UC.LT.2	Service launcher shall send launch request to the necessary services along with application launch.
RQ.LT.5		UC.LT.3	Services that were running when the system was last shut down shall be launched in priority.
RQ.LT.6	Service termination	UC.LT.4	Service terminator shall send a termination request to the service that provided functionality to the application when the application is no longer in use and there is a request to terminate the service.
RQ.LT.7		UC.LT.5	Service terminator shall send a termination request to the necessary service according to a power state transition request.
RQ.LT.8		UC.LT.6	Service terminator shall be able to set the order in which services are terminated during system shutdown.

4.2.4. Service Launch and Termination in Basesystem

4.2.4.1. Reference implementation in Basesystem

In the implementation of Basesystem, the function modules for launching and terminating resident services and non-resident services are separated. In Basesystem implementation, System manager launches and terminates resident services and Task manager launches and terminates non-resident services. As shown in the following figure 14, the Service launcher corresponds to (i) Systemmanager and (ii) Task manager for each trigger in the Basesystem module. (i) When IVI starts, System manager launches services according to the configured order. It needs to be launched first in the IVI system and it is responsible for launching and terminating other services. When IVI shuts down, it terminates the services in the order according to the settings in the same way as at startup. (ii) Task manager launches and terminates the non-resident service.This function is provided as a process and launched by System manager. Task manager provides the service launch interface to HMI application and launches the service if requested. Task manager monitors the running status of the launched service, and if it detects a hang-up, it forcibly terminates the service.

Figure 14



4.2.4.1.1. System manager

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/system/system_manager;hb=refs/heads/master

4.2.4.1.2. Task manager

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/system/task_manager;h=0d795ded146125a0f28b363e8e0821210d1be209;hb=refs/heads/master

4.3. System Failure Detection and Procedures

4.3.1. Abstraction

If the IVI system should detect some kind of failure and determine that the system can no longer maintain normal state, simply rebooting the system and waiting for it to recover is not enough in terms of convenience and safety for users. This is because freezing the screen for a few seconds while driving and just waiting for the system to reboot can cause a very dangerous situation for the user.

Therefore, when the IVI system detects such failure, a recovery procedure needs to be performed. For example, restarting the service that caused the problem, or restarting the entire system. They will prevent the system from continuing in an abnormal state and minimize the negative impact on users.

The following figure shows the roles of the modules that perform detection when a failure is detected and the data flow diagram. Figure 15 shows a case where a failure such as a service hang-up occurs, and Figure 16 shows a case where a failure such as a shortage of resources such as memory occurs. In the IVI system, each service in the IVI system needs to be monitored by heartbeat communication, etc. The Detector monitors the service and when it detects a failure, it notifies the service launcher of the information and sends a request to the service to restart or to restart the entire IVI system to bring the system back to a normal state. System resources(in this case,

Memory, CPU, and GPU) need to be monitored as well. If the Detector monitors the resources and detects a failure, it will take the same steps to recover.

This chapter describes the use cases with failure detection service, the functional requirements to realize the use cases, and the current Basesystem design and implementation as a reference.

Figure 15

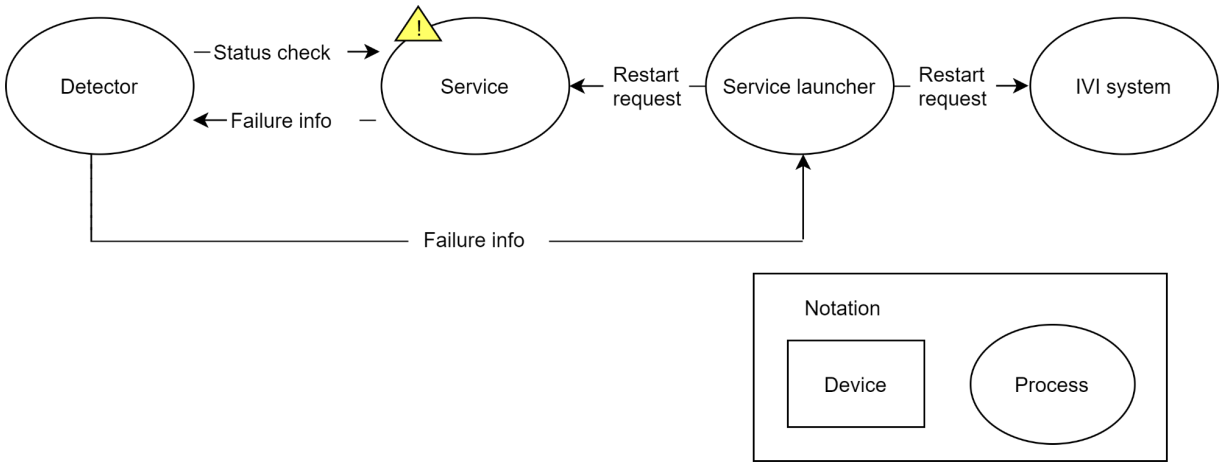
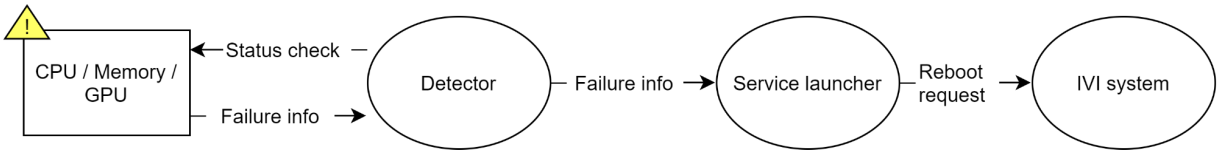


Figure 16



4.3.2. Use cases

In the Table 17, use cases which need System Failure Detection and Procedure when the driver or passenger operates the IVI system are described.

The use cases UC.FD.1 to UC.FD.4 are for the passenger to face the navigation app failures, and UC.FD.5 is for OEM to analyze the failure at a service station later.

Table 17

#	Item	Description
UC.FD.1	Service failure at System startup	The passenger is not able to see the map image on the screen, e.g. the map service cannot be activated.
UC.FD.2	Service failure when System is in use	The passenger is not able to see the map image on the screen due to the route calculation, guidance services, etc. not responding.
UC.FD.3	System memory shortage detection	The map image on the screen has freezed and not been updated due to a shortage of system memory.
UC.FD.4	CPU/GPU high load detection	The navigation map app does not respond in expected time and shows intermittent image updates due to very high work-load of system resources.
UC.FD.5	CPU/GPU usage log	In case of poor usability, i.e. intermittent screen updates, the IVI system resource information is recorded for OEM to analyze the issues later.

4.3.2.1. Functional Requirements

The Table 18 includes the functional requirements of Service Failure Detection and Procedure module. It is assumed that the targets of failure detection are Services, system memory resources, CPU work-load, GPU work-load.

Table 18

#	Item	Related use case	Description

RQ.FD.1	Service failure detection	UC.FD.1, UC.FD.2	The detector shall monitor IVI service health status. If an IVI service does not respond, it shall be recognized the service is in failure status.
RQ.FD.2	Timeout parameter for service monitoring	UC.FD.1, UC.FD.2	The timeout parameter shall be configurable for the detector to wait for the response from an IVI service.
RQ.FD.3	Frequency for service monitoring	UC.FD.1, UC.FD.2	The frequency with which the Detector checks the service should be configurable.
RQ.FD.4	Memory failure detection	UC.FD.3	The detector shall decide the system is in failure status if the system memory consumption exceeds the threshold for the specified periods.
RQ.FD.5	CPU / GPU failure detection	UC.FD.4,	The detector shall decide the system is in failure status if the GPU work-load exceeds the threshold for the specified periods.
RQ.FD.6	Resource failure detection periods for Memory / CPU / GPU usage	UC.FD.3, UC.FD.4	The periods to detect the system resource failure in RQ.FD.3 and RQ.FD.4 shall be configurable.
RQ.FD.7	System/service recovery	UC.FD.1, UC.FD.2, UC.FD.3, UC.FD.4	In case of RQ.FD.1, the detector shall perform the system/service recovery operation, if it detects any system failure.
RQ.FD.8	Logging of failure	UC.FD.5	The detector shall record the diagnostic information, e.g. process information, if it detects CPU/GPU resource failure.

RQ.FD.9	Immediate service shutdown	UC.FD.1, UC.FD.2, UC.FD.3, UC.FD.4	The detector shall notify immediate shutdown of the failure service to the terminator if it detects any service failure.

4.3.3. Service Failure detection in Basesystem

4.3.3.1. Reference implementation in Basesystem

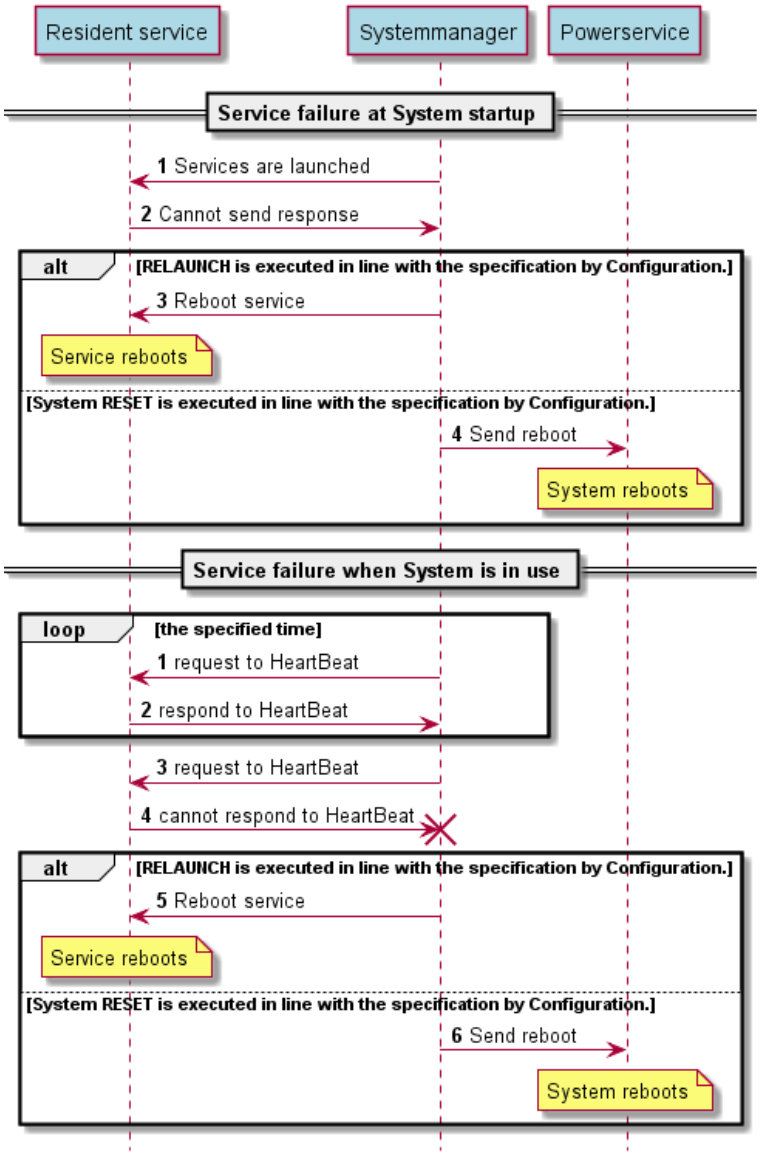
In the implementation of Basesystem, the function modules for Service Failure detection are System manager and Resource manager.

When System manager detects failures on services, it executes the various failure procedure. The contents of the failure procedure are statically prescribed in the Configuration file in advance. The prescribed ones are the system restart and the restart the process in which the failure occured.

System manager monitors the system memory in cooperation with Resource manager. If a notification is received from Resource manager, it recognizes the system memory shortage and resets the system.

The following sequence diagram(Figure 19) shows the sequence of events when a failure occurs in the system as a sample.

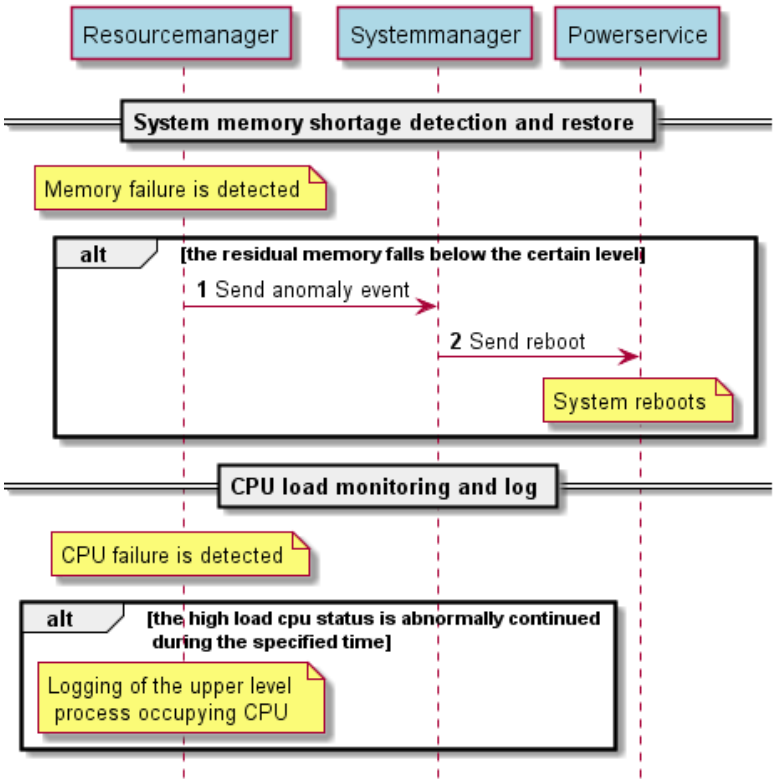
Figure 19



Resource manager is a function that checks the status of the CPU and keeps a log of the high-level processes occupying the CPU if the high-load status continues for a certain period of time. It also checks the status of memory, and if the residual memory gets lower than a certain level, it determines it to be an abnormal state and notifies the System manager.

The following sequence diagram (Figure 19) shows, as a sample, the sequence of events when a memory or CPU failure occurs.

Figure 20



4.3.3.2. System manager

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/system/system_manager;hb=refs/heads/master

4.3.3.3. Resource manager

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/system/resource_manager;h=ab341007c7257d839fb6b91b02444675b9de6d60;hb=refs/heads/master

4.4. System Logging Support

4.4.1. Abstraction

Log and diagnostics information is very important to check the system health condition and analyze potential and already observed issues. As a way of logging, printf() style easy approach to put logging messages can be convenient for developers, however the console is usually not available. Therefore, how and where to save the log need to be considered.

Depending on the use cases to output, collect and analyze log information, it is necessary to prioritize the log-levels. Critical error information must be recorded and shall not be overwritten by the limited log storage on the IVI system. Information such as service or messages may be useful for developers to analyze problems.

This chapter describes the use cases with System Logging Support(the function of logging as described above), the functional requirements for realizing the use cases, and the functions of the Basesystem that can be used as a sample implementation.

4.4.2. Use cases

In the Table 21, use cases which need System Logging Support module are described.

Table 21

#	Item	Description
UC.LS.1	Check logs during development	The developers (OEM/Supplier) check and evaluate the logs by each function (application) when implementing the product software.
UC.LS.2	Check logs of already used products	In order to investigate the status and problems of used products, OEM analyze the stored logs by each function (application).

4.4.3. Functional Requirements

The Table 22 includes the functional requirements of System Logging Support module.

Table 22

#	Item	Related use case	Description
RQ.LS.1	Log save	UC.LS.2	The System Logging Support module shall preserve any recorded log messages even through system startup/shutdown lifecycle.
RQ.LS.2		UC.LS.1, UC.LS.2	The storage of logs shall be able to be configurable(e.g. The log storage device, directory or cloud).
RQ.LS.3		UC.LS.1, UC.LS.2	The logs shall be compressed when they are saved.
RQ.LS.4	The setting of log messages	UC.LS.1, UC.LS.2	Log messages should have several levels depending on their importance. For example, the following. ● Error: Information when some errors happen ● Info: Information which is not error

			● Debug: Information needed when debug for development
RQ.LS.5		UC.LS.1, UC.LS.2	The log-level of the log messages shall be configurable.
RQ.LS.6		UC.LS.1, UC.LS.2	The log messages shall be categorized according to the functionalities or domains.
RQ.LS.7		UC.LS.1, UC.LS.2	The System Logging Support module shall record the timestamp of the log messages.
RQ.LS.8		UC.LS.1, UC.LS.2	The System Logging Support module shall not overwrite any critical log information, e.g. information on possible reasons to restart the system, even though the system log storage is full.
RQ.LS.9	Storage memory	UC.LS.1, UC.LS.2	The System Logging Support module shall minimize actual write operations to the storage device to make the lifetime as long as possible.

4.4.4. System Logging Support in Basesystem

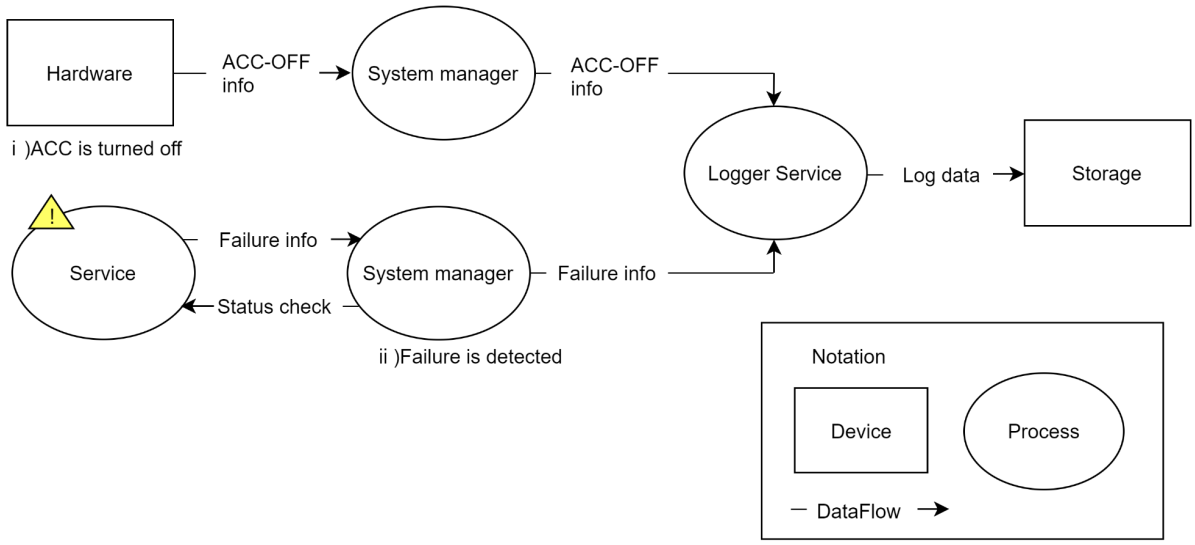
4.4.4.1. Reference implementation in Basesystem

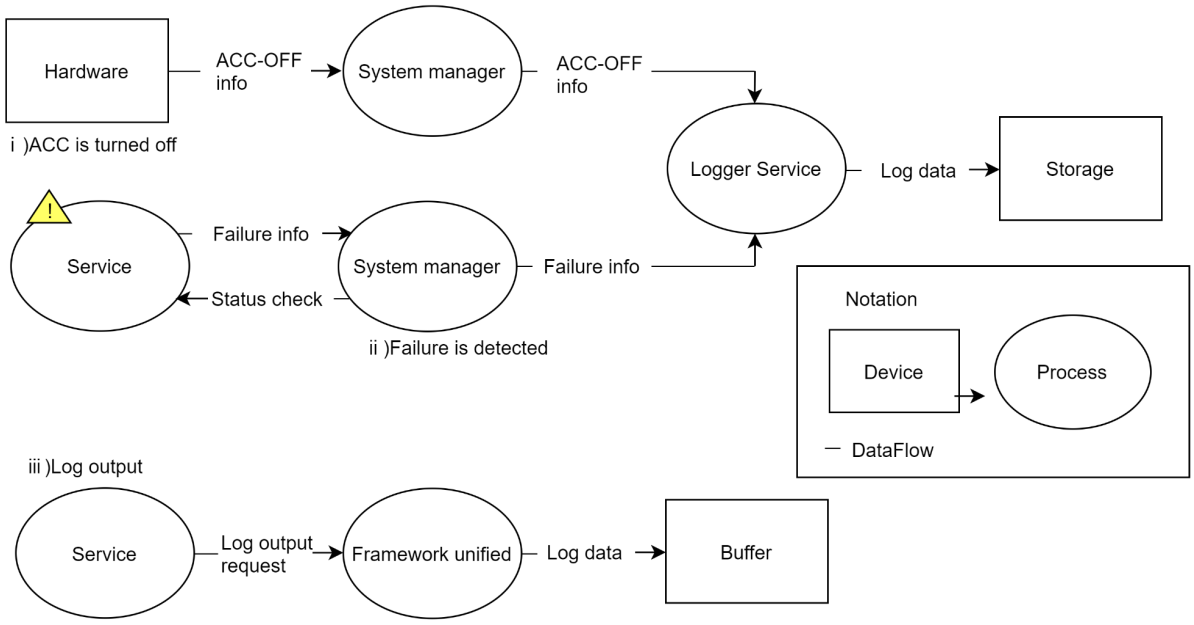
In the implementation of Basesystem, the function module for System Logging Support is partially covered by Logger Service and Framework Unified.

Logger Service does not create logs by itself, but rather collects and archives the logs left by other modules when there is a request to save them in the system. As shown in Fig. 24, for example, when the ACC is turned off(ⅰ), the system manager notifies the Logger Service of this information, and the Logger Service collects the logs from other modules covering the period from system startup to shutdown, and stores them in the storage. When an failure occurs in the system(ⅱ), it is another trigger for logging, and the Logger Service stores the log by receiving the request. The logs are stored in the

volatile area on a regular basis, but when the capacity becomes full, they are stored in the non-volatile area. Framework Unified is responsible for outputting the logs. Upon request from the required service, a log level is specified and the log is output to a buffer(iii). Logging levels include typical definitions such as Info, Warning, and Error, and additional levels can be set optionally by the user. These functions are the role of System Logging Support.

Figure 24





4.4.4.1.1. Logger Service

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/system/logger_service;h=3b33fe43aa5914210ac1e3e485d118a9d8d103bd;hb=refs/heads/master

4.4.4.1.2. Framework Unified

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/native/framework_unified;h=a1c0913dabd4d65c6c2d4dd90646e20ea9977ffe;hb=refs/heads/master

4.5. Persistent Data Management

4.5.1. Abstraction

While the IVI system is running, user configuration data and IVI service operation data are stored and maintained. Even though IVI is battery-powered which looks similar to smartphones, electronic consumer devices, the power supply in IVI system can be unstable at ACC-ON and ACC-OFF for the system startup/shutdown. Therefore, the Persistent Data Management module must ensure that no persistent data is lost, no persistent data is corrupted and the persistent data is consistent while the system startup/shutdown lifecycle.

Another aspect on IVI system which is special rather than smartphone, electronic consumer devices is a much longer product lifetime, and that means the storage device must be available for a much longer time. So, in order to make the storage device lifetime longer, data write operation shall not happen every time when applications request. The Persistent Data Management module needs to handle the read/write operation to the storage device at any given time.

This chapter describes the use cases with Persistent Data Management(the function as described above), the functional requirements for realizing the use cases, and the functions of the Basesystem that can be used as a sample implementation.

4.5.2. Use cases

In the Table 25, use cases which need the Persistent Data Management module for services are described.

Table 25

#	Item	Description
UC.BD.1	Data protection in case of ACC-OFF	Even when the driver presses the button for ACC-OFF or a sudden power failure occurs after

	or sudden power failure	changing the display settings of the IVI system, the display settings are retained and the driver uses the information.
UC.BD.2	Utilize the persistent data	The driver searches for a destination using the search history stored in the navigation application.
UC.BD.3	Data attribute when the system battery is removed	Each OEM chooses whether to make each persistent data stored in the IVI volatile or non-volatile when the battery is removed, for handling the persistent data after the battery is removed.
UC.BD.4	Data Initialization	If a user reset the system, all data is deleted.

4.5.3. Functional Requirements

The Table 26 includes the functional requirements of Persistent Data Management module. RQ.BD.1.1 and RQ.BD.1.2 are requirements realized by usual filesystem. Others are newly defined.

Table 26

#	Item	Related use case	Description
RQ.BD.1.1	Data store in case of sudden power off	UC.BD.1	The Persistent Data Management module shall prevent “persistent” data loss if the power supply is stopped.
RQ.BD.1.2	Data store of system lifecycle	UC.BD.1	IVI system shall preserve any "persistent" data requested by IVI services/apps through system startup/shutdown lifecycle.

RQ.BD.2.1	Storage memory requirement	UC.BD.1	The Persistent Data Management module shall minimize actual write operations to the storage device to make the lifetime as long as possible.
RQ.BD.2.2	Utilization of Persistent data	UC.BD.2	The data which needs to be read shall be readable by request from IVI services/apps.
RQ.BD.2.3	Data handling when the battery is removed.	UC.BD.3	Data shall be configurable to be kept or not when the battery is removed.
RQ.BD.2.4	Data Verification	UC.BD.2	The Persistent Data Management module shall detect the stored data corruption and provide alternative correct data.
RQ.BD.2.5	Data deletion	UC.BD.4	The data shall be deleted by request from IVI services/apps.

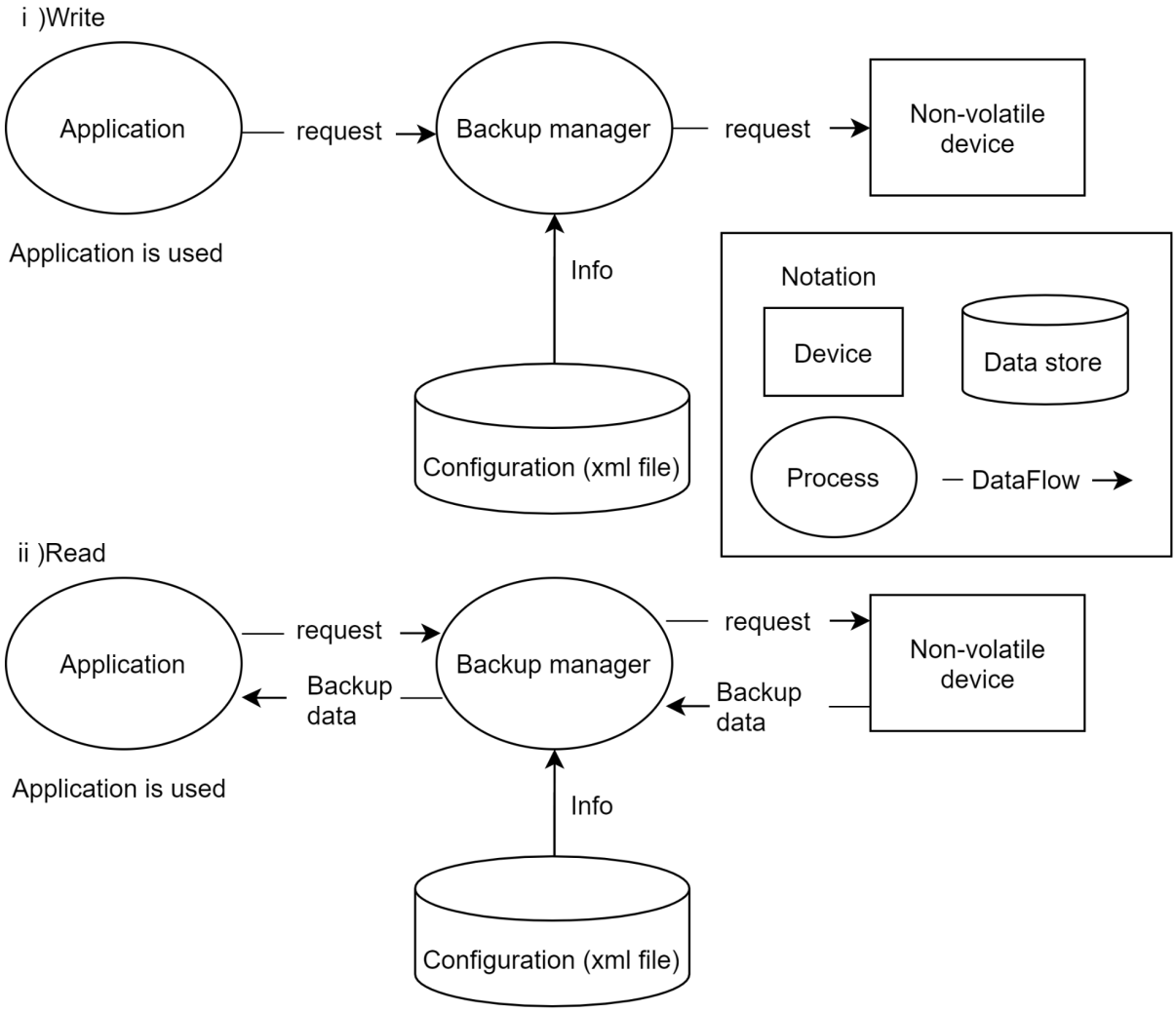
4.5.4. Persistent Data Management in Basesystem

4.5.4.1. Reference implementation in Basesystem

In the implementation of Basesystem, the function module for Persistent Data Management is Backup Manager.

As shown in the following figure 27, for example when a driver uses an application and a data backup request occurs, on receiving the request, Backup Manager writes the data to the specified storage with the specified offset and data size based on the timing defined in the Configuration(ⅰ). When reading the data, Backup manager reads the data from the specified storage and sends it back to the application side(ⅱ). Backup Manager has responsible to provide those features. It will manage not only access to the persistent data for applications but also verify the underlying persistent data consistency in order to detect the data corruption so that alternative correct data can be provided. It also can delete part of the persistent data upon the requests from applications.

Figure 27



4.5.4.1.1. Backup manager

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/native/backup_manager;h=2a9382f9cbf84a8a2f3e1cb4602a693f44bd37d7;hb=refs/heads/master

4.6. Vehicle Parameter Configuration

4.6.1. Abstraction

IVI system requires software that is designed to meet a variety of needs and requirements. These needs and requirements vary depending on the country where it is used. In addition, OEMs want to provide multiple models and grades of vehicle, but their requirements also vary. In order to prevent huge numbers of IVI system software branching, a configuration mechanism is needed to cover several options without changing the program. This will be accomplished with this Vehicle Parameter Configuration. The information obtained from this function will allow, for example, the IVI system to set the required language for different countries and to select the Audio AMP that the product supports. In addition, based on the vehicle parameter, it determines if the vehicle is compatible with OEM-specific or country-specific features and returns it to the service.

This chapter describes the use cases with Vehicle Parameter Configuration, the functional requirements for realizing the use cases, and the functions of the Basesystem that can be used as a sample implementation.

4.6.2. Use cases

In the Table 28, use cases which need the Vehicle Parameter Configuration module for services are described.

Table 28

#	Item	Description
UC.VP.1	Vehicle parameter setting before shipment	Before the product is shipped from the factory, each OEM/Supplier writes the configuration values into the product depending on the needs and requirements of the product without changing the software.

UC.VP.2	Function check	When the driver presses the button for ACC-ON and the IVI system is started, the system obtains information on which functions the product supports from the vehicle parameter that has been set. For example, functions specific to communication standards, devices, applications, etc. are enabled/disabled based on the vehicle parameter.

4.6.3. Function Requirements

The Table 29 includes the functional requirements of Vehicle Parameter Configuration module.

Table 29

#	Item	Related use case	Description
RQ.VP.1	Vehicle parameter setting	UC.VP.1	This function shall provide at least the following information as vehicle parameters ● Country / Region ● Vehicle type/brand ● Vehicle signal acquisition method (e.g.CAN or direct)
RQ.VP.2	Acquisition of vehicle parameter	UC.VP.1	If a request is received from a service that requires a vehicle parameter, Vehicle Parameter Configuration module shall provide the necessary information.
RQ.VP.3	Function check	UC.VP.2	Vehicle Parameter Configuration module determines information on whether the product supports various specific functions and returns whether the functions can be used or not.

4.6.4. Vehicle Parameter Configuration in Basesystem

4.6.4.1. Reference implementation in Basesystem

In the implementation of Basesystem, the function module for Vehicle Parameter Support is Vehicle Parameter Library.

In the implementation of Basesystem, it is assumed that the vehicle parameter information is stored and set in the IVI system as the configuration file. A service specifies the vehicle parameter information to acquire and sends a request to Vehicle Parameter Library, and Vehicle Parameter Library reads the data from the configuration file of vehicle parameter and stores it in the specified address.If the value of the specified variable is not set in the vehicle parameter's configuration file, 0 is returned.

In addition, when the service sends a request to the Vehicle Parameter Library to obtain information on whether the product supports various functions (communication standards, devices, applications, etc.), the Vehicle Parameter Library returns the result of enabled, disabled, or function not existing.

4.6.4.2. Vehicle Parameter Library

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/other/vehicle_parameter_library;h=021a98c97e1a4ab0402595da3f6c4db6e34fd5ba;hb=refs/heads/master

4.7. CAN Communication

4.7.1. Abstraction

CAN is the most major communication bus among ECUs in-vehicle systems. It is sufficient and reliable to obtain vehicle information such as speed data and HVAC control commands.

Linux has SocketCAN implementation for CAN communication to utilize the popular Berkley socket API. However, on an IVI system, it is expected that a communication function will be provided to meet the following requirements.

- The IVI platform needs to notify CAN data from other ECUs or from external sources to the necessary applications which request to receive the CAN data.
- The IVI platform needs to monitor the periodic reception of CAN data from external sources and to notify the availability of delivery

For these reasons, it is necessary to define a communication function for handling CAN data.

This chapter describes the use cases with CAN Communication function, the functional requirements for realizing the use cases, and the functions of the Basesystem that can be used as a sample implementation.

4.7.2. Use cases

In the Table 30, use cases which need GPS and Vehicle Sensor Communication module are described.

Table 30

#	Item	Description

UC.CC.1	CAN data send from application to other ECU	When the user presses the button to lower the temperature of the air conditioner, a request is sent to the ECU that controls the temperature of the air conditioner. This causes cold air to come out to lower the interior temperature.
UC.CC.2	CAN data distribution to registered application	IVI system receives the signal data of the steering sensor via CAN bus, when a user is driving the car to park with the gear set reverse checking the predicted trajectory line on the Back-guide monitor.
UC.CC.3	Detect suspension and resume	The CAN communication module receives the CAN data and notifies the registered destination of the data. If the required data is not received for a certain period of time, an unreceived event is sent to the destination, and if the data is received, the destination is notified of the data.

4.7.3. Function Requirements

The Table 31 includes the functional requirements of CAN Communication module.

Table 31

#	Item	Related use case	Description
RQ.CC.1	Utilization of the function from the application	UC.CC.1	CAN Communication module shall receive the CAN data send request from the application.
RQ.CC.2	CAN data distribution registration	UC.CC.2	CAN Communication module shall be able to register the distribution of applications that want to receive CAN data.

RQ.CC.3		UC.CC.2	CAN Communication module shall send CAN data to the registered application.
RQ.CC.4	Detect suspension	UC.CC.3	CAN Communication module shall notify the registered delivery destination of not being received if CAN data is not received within the specified time.
RQ.CC.5	Resume after suspension	UC.CC.3	In the case of RQ.CC.4 status, if CAN Communication module receives CAN data, it shall notify the registered application of the CAN data.

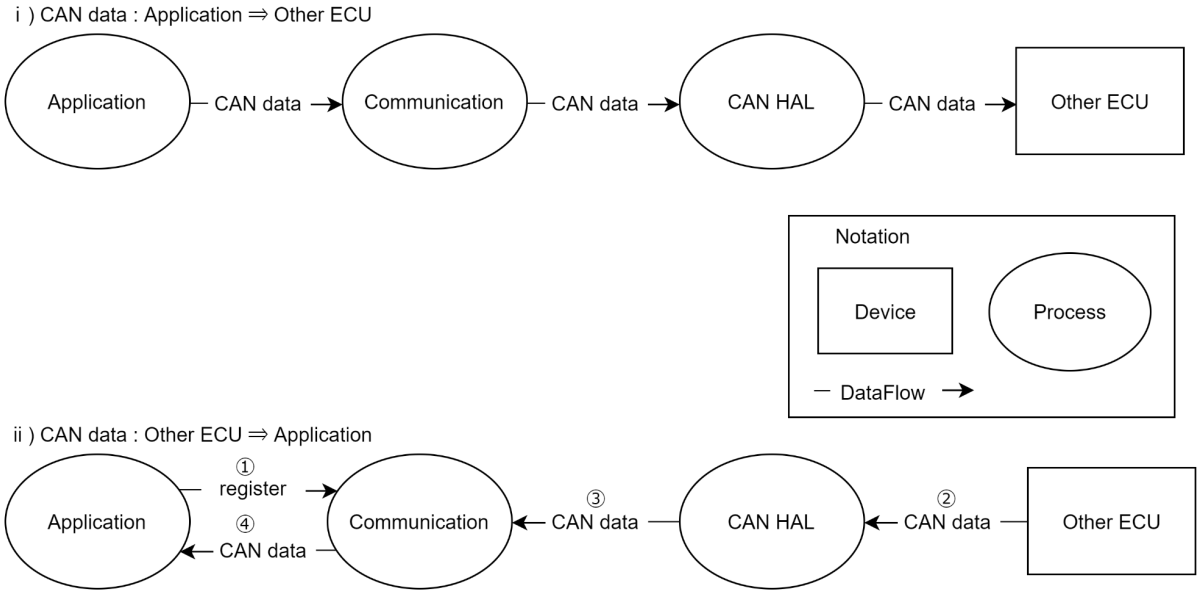
4.7.4. CAN Communication in Basesystem

4.7.4.1. Reference implementation in Basesystem

In the Basesystem implementation, the functional module that performs CAN communication is Communication. The CAN data flow based on UC.CC.1 and UC.CC.2 of the use case is illustrated in Figure 32. In the Basesystem implementation, the CAN communication functions are implemented via the HAL, which is implemented to allow data exchange independent of the device.

- Send CAN data received from the application to CAN HAL.
- Send CAN data received from CAN HAL to the registered applications.
- Monitor the CAN data communication suspension and notify the registered applications of resume.

Figure32



The contributed CAN HAL is a Stub with no implementation part, and if you want to use the Communication module, you have to implement the HAL.

4.7.4.1.1. Communication

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/peripheral/communication;h=c618e7c98854a37d678da03828d4d516173d2b0f;hb=refs/heads/master

4.8. GPS and Sensor information

4.8.1. Abstraction

For IVI system, Positioning information is a very essential one as it is moving. Positioning information includes a variety of information. For example, it includes the

vehicle's longitude and latitude position coordinates obtained from the GPS signal, as well as the vehicle's direction, speed data. These information shall be provided by the IVI platform. Additionally the IVI system has the most intelligent and powerful computing resources ever and geographical information of map data. By acquiring and using Sensor data such as gyroscopes, acceleration, and vehicle speed pulses, IVI platform is expected to provide coordinated location information with dead reckoning navigation and map matching to the required applications. With these backgrounds, IVI system needs to provide functions that can provide GPS and Sensor data to necessary applications.

This chapter describes the use cases with GPS and Sensor communication module, the functional requirements for realizing the use cases, and the functions of the Basesystem that can be used as a sample implementation.

4.8.2. Use cases

In the Table 33, use cases which need GPS and Sensor Communication module are described.

Table 33

#	Item	Description
UC.GS.1	Utilization of current location information	The driver uses the navigation application and sets the destination. The user starts the application, enters the destination, and the time required and route are displayed.
UC.GS.2	Acquisition of GPS time	IVI system obtains the GPS time and corrects the display of the time on the IVI display screen. It can also be used for other way as "expiration date" for credentials by openssl or abs time of linux, etc. as examples.
UC.GS.3	DeadReckoning	A driver drives a car to a destination. If the driver enters a tunnel or other place where it is difficult to receive a signal and GPS information cannot be

		obtained, the driver can use the Sensor information obtained to position the car.
UC.GS.4	Retention of GPS information	When the driver presses the button for ACC-OFF, the IVI system retains the GPS information. When the driver presses the button for ACC-ON, the current location information will be displayed based on the retained information.

4.8.3. Function Requirements

The Table 34 includes the functional requirements of GPS and Sensor Communication module.

Figure 34

#	Item	Related use case	Description
RQ.GS.1	GPS and Sensor data distribution registration	UC.GS.1, UC.GS.2	GPS and Sensor communication module shall be able to register the delivery of GPS(including time) and Sensor data to the application that wants to receive them.
RQ.GS.2	Utilization of the function from the application	UC.GS.1, UC.GS.2	GPS and Sensor communication module shall send the necessary GPS(including time) and Sensor data to the registered applications.
RQ.GS.3	Acquisition of GPS data and Sensor data	US.GS.3	GPS and Sensor communication module shall receive GPS and Sensor data transmission requests from applications. Dead Reckoning function is not provided by this system, and needs to be implemented

			separately. Since acquired GPS and Sensor data, etc. are required, the data will be provided when the request is sent from the system owning Dead Reckoning function.
RQ.GS.4	Provision of GPS data and Sensor data	UC.GS.3	GPS and Sensor communication module shall receive the GPS and Sensor data transmission request from the application and provide the necessary data.
RQ.GS.5	Retention of GPS information	UC.GS.4	GPS and Sensor communication module retains GPS information when the ACC is turned OFF from ON, and provides GPS information when the ACC is turned ON.

4.8.4. GPS and Sensor information in Basesystem

4.8.4.1. Reference implementation in Basesystem

In the Basesystem implementation, the functional module that handles GPS and Sensor data is Positioning. Positioning provides the following functions.

- Provide location information (longitude, latitude, altitude, heading) and speed information to the required application/service.
- Provide GPS data and GPS time to required applications/services, and reset GPS.
- Provide Sensor data (gyroscope, acceleration, vehicle speed pulse, vehicle reverse information, etc.)

Application/Service requests GPS and Sensor data from Positioning, and returns the information obtained from Vehicle.

4.8.4.1.1. Positioning

https://gerrit.automotivelinux.org/gerrit/gitweb?p=staging/basesystem.git;a=tree;f=service/vehicle/positioning;h=6573a2b89dc948502838480dcbaae18995f29fd4;hb=refs/heads/master