

<b>Document Title</b>	Specification of Vehicle Update and Configuration Management
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	1090

<b>Document Status</b>	published
<b>Part of AUTOSAR Standard</b>	Adaptive Platform
<b>Part of Standard Release</b>	R23-11

<b>Document Change History</b>			
<b>Date</b>	<b>Release</b>	<b>Changed by</b>	<b>Description</b>
2023-11-23	R23-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Initial release from split of SWS UCM document</li> <li>• Vehicle dependency handling</li> <li>• VSM interface refactoring</li> </ul>

## Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

## Contents

1	Introduction and functional overview	6
2	Acronyms and Abbreviations	7
3	Related documentation	8
3.1	Input documents & related standards and norms	8
3.2	Further applicable specification	8
4	Constraints and assumptions	9
4.1	Known limitations	9
4.2	Applicability to car domains	9
5	Dependencies to other Functional Clusters	10
5.1	Provided Interfaces	10
5.2	Required Interfaces	10
6	Requirements Tracing	13
7	Functional specification	15
7.1	Functional cluster life-cycle	15
7.2	Startup	15
7.3	Shutdown	15
7.4	Technical Overview	15
7.5	V-UCM general behavior	17
7.6	UCM identification	17
7.7	V-UCM Software Packages transfer or streaming	18
7.8	V-UCM retry strategy	19
7.9	Adaptive Applications interacting with V-UCM	20
7.9.1	OTA Client	20
7.9.2	Vehicle Driver Interface	22
7.9.3	Vehicle State Manager	23
7.9.4	Flashing Adapter	25
7.9.5	UCM Diagnostic Application	25
7.9.6	Non Adaptive Platform update	27
7.9.6.1	D-PDU API implementation support	27
7.9.6.2	Not required D-PDU API concepts	28
7.9.6.3	Not required D-PDU API functions	28
7.9.6.4	Classic platform update with V-UCM and diagnostic tool	30
7.10	Status reporting	31
7.10.1	States	33
7.10.2	States Transitions	34
7.11	Campaign cancelling	37
7.12	Campaign Reporting	38
7.13	Content of Vehicle Package	39

7.14	Vehicle update security and confidentiality	41
8	API specification	42
9	Service Interfaces	43
9.1	Type definitions	43
9.1.1	CampaignHistoryType	45
9.1.2	CampaignHistoryVectorType	46
9.1.3	CampaignResultType	46
9.1.4	UCMMasterResolutionVectorType	46
9.1.5	UCMMasterResolutionType	47
9.1.6	UCMStepErrorVectorType	47
9.1.7	UCMStepErrorType	48
9.1.8	SoftwarePackageStepType	48
9.1.9	UCMHistoryType	48
9.1.10	UCMHistoryVectorType	49
9.1.11	CampaignStateType	49
9.1.12	TransferStateType	50
9.1.13	SafetyConditionType	50
9.1.14	VehicleConditionCollectionType	50
9.1.15	VehicleConditionType	51
9.1.16	SafetyStateType	51
9.1.17	SwNameVersionVectorType	51
9.1.18	VehicleUCMInfo	52
9.1.19	UCMIdentifiersAndVersionsType	52
9.1.20	SwPackageDescType	52
9.1.21	SwPackageDescVectorType	53
9.2	Provided Service Interfaces	53
9.2.1	Vehicle Package Management	53
9.2.2	Vehicle Driver Application Interface	59
9.2.3	Vehicle State Manager	62
9.3	Required Interface	63
9.4	Application Errors	64
9.4.1	Application Error Domain	64
9.4.1.1	UCMErrorDomain	64
10	Configuration	66
10.1	Default Values	66
10.2	Semantic Constraints	66
A	Mentioned Manifest Elements	67
B	Platform Extension API (normative)	74
C	Interfaces to other Functional Clusters (informative)	75
C.1	Overview	75
C.2	Interface Tables	75

D	Packages distribution within vehicle detailed sequence examples	76
D.1	Collect information of present Software Clusters in vehicle	76
D.2	Action computation	76
D.2.1	Pull package from Backend into vehicle	77
D.2.2	Push package from backend into vehicle	78
D.3	Packages transfer from backend into targeted UCM	79
D.4	Package processing	81
D.5	Package activation	83
D.6	Package rollback	85
D.7	Campaign reporting	86
E	Security Analysis of Installation and Update	87
E.1	Securing Vehicle Package	87
E.2	Securing Calls to V-UCM	87
E.3	Suppressing Call to V-UCM	88
E.4	Resource Starvation	88
F	Demonstrator Examples	89
G	Change history of AUTOSAR traceable items	116
G.1	Traceable item history of this document according to AUTOSAR Release R23-11	116
G.1.1	Added Specification Items in R23-11	116
G.1.2	Changed Specification Items in R23-11	119
G.1.3	Deleted Specification Items in R23-11	119
G.1.4	Added Constraints in R23-11	120
G.1.5	Changed Constraints in R23-11	120
G.1.6	Deleted Constraints in R23-11	120

# 1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR functional cluster [Vehicle Update and Configuration Management](#) which belongs to the [AUTOSAR Adaptive Platform Services](#). One of the declared goals of the [AUTOSAR Adaptive Platform](#) is the ability to flexibly update the software and its configuration through local (“tester-based”) or remote (“over-the-air”) updates. [V-UCM](#) provides services for updating the software and its configuration in a vehicle. [V-UCM](#) is coordinating an update campaign within the vehicle. Therefore this document includes requirements on the following functionalities:

- Interact with [Backend](#) to:
  - Identify [Software Clusters](#) that could be updated, installed or removed
  - Authenticate [Vehicle Package](#)
  - Confirm dependencies between [Software Clusters](#) within vehicle before starting campaign
  - Inform [Backend](#) of needed [Software Packages](#), receives them and dispatch them to targeted [ECUs](#)
- Interact with [Vehicle State Manager](#):
  - Inform which safety conditions that have to be applied according to [Vehicle Package](#)
  - Share the computed vehicle state to other Applications or [Functional Clusters](#) involved in the update campaign
- Interact with Human Driver about update campaign:
  - Provides campaign state to trigger interaction with Human during update campaign
  - Get vehicle modification approval or consent from Human when configured in [Vehicle Package](#)
- Provide information of installed software in vehicle
- Provide information of update campaigns history
- Recovery in case of failure

## 2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the [V-UCM](#) module that are not included in the AUTOSAR TR Glossary.

Abbreviation / Acronym:	Description:
DM	AUTOSAR Adaptive Diagnostic Management
UCM	Update and Configuration Management
V-UCM	V-UCM is distributing packages and coordinating an update campaign in a vehicle (former name was UCM Master)
Backend	Backend is a server hosting Software Packages
OTA Client	OTA Client is an Adaptive Application in communication with Backend Over The Air
Vehicle Driver Application	Vehicle Driver Application is an Adaptive Application in communication with Vehicle Driver Human to Machine Interface
Application Error	Errors returned by UCM
Boot options	Boot Manager Configuration
VCI	Vehicle Communication Interface
MVCI	Modular Vehicle Communication Interface
D-PDU API	Diagnostic Protocol Data Unit Application Programming Interface
RDF	Root Description File
MDF	Module Description File
integrity check	verification method proving there has not been any alteration of the artefact content
dependency check	verification method proving that all configured dependencies will be fulfilled after finishing the activation.

Below acronyms and abbreviations relevant for this document are included in the AUTOSAR TR Glossary. This is to avoid duplicate definition of the technical term. And to refer to the correct document.

Term	Description:
Adaptive Application	see AUTOSAR TR Glossary
AUTOSAR Adaptive Platform	see AUTOSAR TR Glossary
AUTOSAR Classic Platform	see AUTOSAR TR Glossary
Functional Cluster	see AUTOSAR TR Glossary
Service	see AUTOSAR TR Glossary
Electronic Control Unit	see AUTOSAR TR Glossary
Machine	see AUTOSAR TR Glossary
Manifest	see AUTOSAR TR Glossary
Software Package	see AUTOSAR TR Glossary
Software Cluster	see AUTOSAR TR Glossary
Vehicle Package	see AUTOSAR TR Glossary
Vehicle State Manager	see [1] AUTOSAR Glossary

**Table 2.1: Reference to Technical Terms**

## 3 Related documentation

### 3.1 Input documents & related standards and norms

- [1] Glossary  
AUTOSAR\_FO\_TR\_Glossary
- [2] General Requirements specific to Adaptive Platform  
AUTOSAR\_AP\_RS\_General
- [3] Explanation of Adaptive Platform Software Architecture  
AUTOSAR\_AP\_EXP\_SWArchitecture
- [4] Requirements on Vehicle Update and Configuration Management  
AUTOSAR\_AP\_RS\_VehicleUpdateAndConfigurationManagement
- [5] Specification of Update and Configuration Management  
AUTOSAR\_AP\_SWS\_UpdateAndConfigurationManagement
- [6] Explanation of Adaptive Platform Design  
AUTOSAR\_AP\_EXP\_PlatformDesign
- [7] Specification of Manifest  
AUTOSAR\_AP\_TPS\_ManifestSpecification

### 3.2 Further applicable specification

AUTOSAR provides a general specification [2] which is also applicable for [V-UCM](#). The specification RS General shall be considered as additional and required specification for implementation of [V-UCM](#).



## **4 Constraints and assumptions**

### **4.1 Known limitations**

No limitations

### **4.2 Applicability to car domains**

No restrictions to applicability.

## 5 Dependencies to other Functional Clusters

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 “Provided Interfaces” lists the interfaces provided by Vehicle Update and Configuration Management to other Functional Clusters. Section 5.2 “Required Interfaces” lists the interfaces required by Vehicle Update and Configuration Management.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [3].

### 5.1 Provided Interfaces

Interface	Functional Cluster	Purpose
No provided interfaces		

Table 5.1: Interfaces provided to other Functional Clusters

### 5.2 Required Interfaces

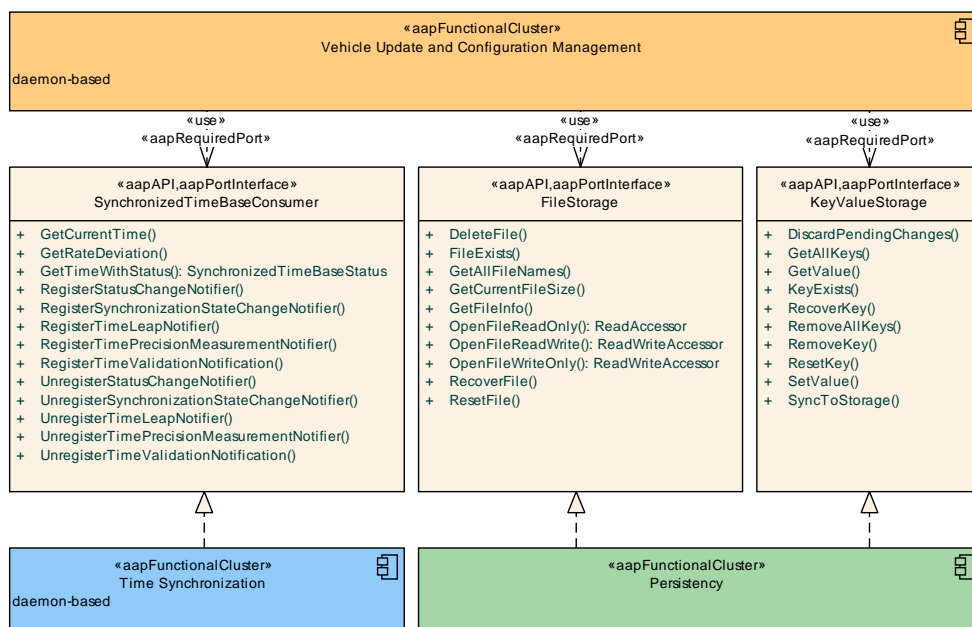
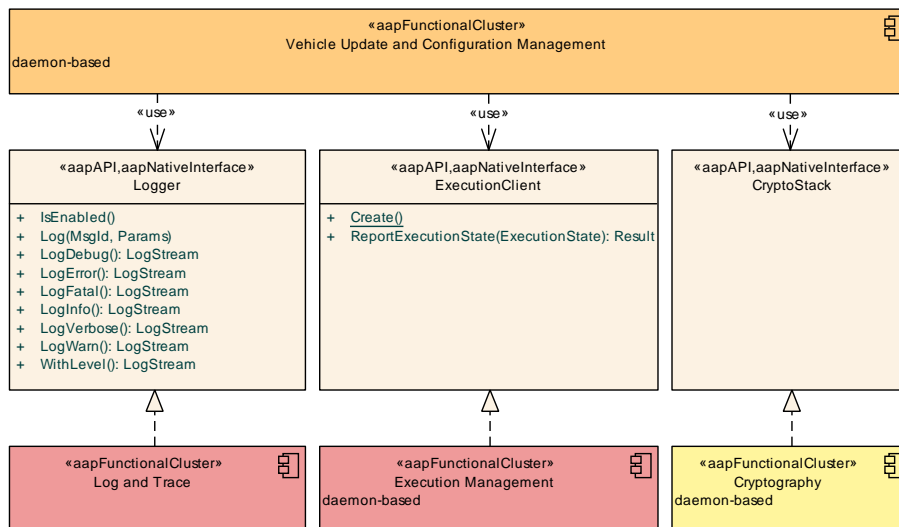
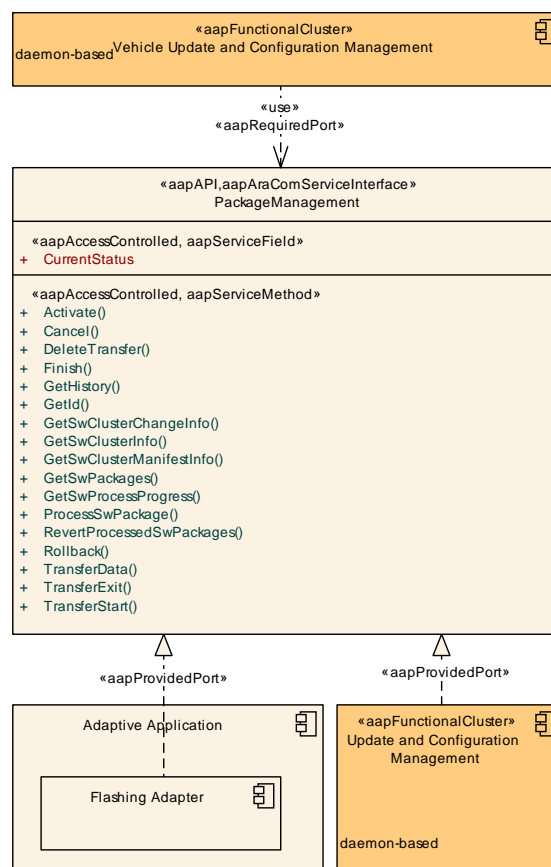


Figure 5.1: Interfaces required by Vehicle Update and Configuration Management from other Functional Clusters



**Figure 5.2: Interfaces required by Vehicle Update and Configuration Management from other Functional Clusters**

Figures 5.1 and 5.2 show the interfaces required by Vehicle Update and Configuration Management from other Functional Clusters within the AUTOSAR Adaptive Platform.



**Figure 5.3: Interfaces required by Vehicle Update and Configuration Management from Update and Configuration Management**

Figure 5.3 shows the interfaces required by Vehicle Update and Configuration Management from Update and Configuration Management.

<b>Functional Cluster</b>	<b>Interface</b>	<b>Purpose</b>
Cryptography	CryptoStack	This interface may be used e.g., to verify the integrity and authenticity of <code>Vehicle Packages</code> .
Execution Management	ExecutionClient	This interface shall be used by the daemon process(es) inside <code>Vehicle Update and Configuration Management</code> to report their execution state to <code>Execution Management</code> .
Log and Trace	Logger	<code>Vehicle Update and Configuration Management</code> shall use this interface to log standardized messages.
Persistency	FileStorageOperations	<code>Vehicle Update and Configuration Management</code> should use this interface to store files, for example received software packages.
Persistency	KeyValueStorageOperations	<code>Vehicle Update and Configuration Management</code> should use this interface to store its internal state.
Persistency	FileStorage	<code>Vehicle Update and Configuration Management</code> should use this interface to store files, for example received software packages.
Persistency	KeyValueStorage	<code>Vehicle Update and Configuration Management</code> should use this interface to store its internal state.
Persistency	ReadAccessor	<code>Vehicle Update and Configuration Management</code> should use this interface to store files, for example received software packages.
Persistency	ReadWriteAccessor	<code>Vehicle Update and Configuration Management</code> should use this interface to store files, for example received software packages.
Time Synchronization	SynchronizedTimeBaseConsumer	<code>Vehicle Update and Configuration Management</code> shall use this interface to get latest timestamp.
Update and Configuration Management	PackageManagement	This interface is used to control different <code>Update and Configuration Management</code> instances and e.g., applications implementing the same interface located within the vehicle that act as an adapter to install software packages on third-party systems. <code>Vehicle Update and Configuration Management</code> is able to differentiate between the service instances by matching the result of <code>GetId</code> with an ID provided in a <code>Software Package</code> .

**Table 5.2: Interfaces required from other Functional Clusters**

## 6 Requirements Tracing

The following tables reference the requirements specified in [4] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_UCM_00013]	UCM shall check that it has enough resources to receive, process and store the Software Package and associated data	[SWS_VUCM_01011]
[RS_VUCM_00033]	V-UCM shall support reporting version information of a complete vehicle	[SWS_VUCM_00181] [SWS_VUCM_00182] [SWS_VUCM_00268] [SWS_VUCM_00269] [SWS_VUCM_01101] [SWS_VUCM_01103] [SWS_VUCM_01120] [SWS_VUCM_01135] [SWS_VUCM_01218] [SWS_VUCM_01311] [SWS_VUCM_CONSTR_00013]
[RS_VUCM_00034]	V-UCM shall record all V-UCM's action history	[SWS_VUCM_00181] [SWS_VUCM_00182] [SWS_VUCM_00251] [SWS_VUCM_00252] [SWS_VUCM_00253] [SWS_VUCM_00254] [SWS_VUCM_00255] [SWS_VUCM_00256] [SWS_VUCM_00290] [SWS_VUCM_00291] [SWS_VUCM_00296] [SWS_VUCM_00304] [SWS_VUCM_01247] [SWS_VUCM_01248] [SWS_VUCM_01266] [SWS_VUCM_01267] [SWS_VUCM_01268] [SWS_VUCM_01269] [SWS_VUCM_01279]
[RS_VUCM_00035]	V-UCM shall coordinate software update in a vehicle across multiple Electronic Control Units	[SWS_VUCM_00178] [SWS_VUCM_00181] [SWS_VUCM_00183] [SWS_VUCM_00210] [SWS_VUCM_00297] [SWS_VUCM_00298] [SWS_VUCM_00307] [SWS_VUCM_00308] [SWS_VUCM_01013] [SWS_VUCM_01018] [SWS_VUCM_01020] [SWS_VUCM_01119] [SWS_VUCM_01122] [SWS_VUCM_01123] [SWS_VUCM_01124] [SWS_VUCM_01131] [SWS_VUCM_01132] [SWS_VUCM_01177] [SWS_VUCM_01178] [SWS_VUCM_01204] [SWS_VUCM_01205] [SWS_VUCM_01207] [SWS_VUCM_01209] [SWS_VUCM_01212] [SWS_VUCM_01214] [SWS_VUCM_01215] [SWS_VUCM_01216] [SWS_VUCM_01217] [SWS_VUCM_01218] [SWS_VUCM_01219] [SWS_VUCM_01220] [SWS_VUCM_01221] [SWS_VUCM_01222] [SWS_VUCM_01227] [SWS_VUCM_01228] [SWS_VUCM_01229] [SWS_VUCM_01234] [SWS_VUCM_01236] [SWS_VUCM_01239] [SWS_VUCM_01240] [SWS_VUCM_01241] [SWS_VUCM_01242] [SWS_VUCM_01243] [SWS_VUCM_01244] [SWS_VUCM_01246] [SWS_VUCM_01270] [SWS_VUCM_01271] [SWS_VUCM_01272] [SWS_VUCM_01273] [SWS_VUCM_01274] [SWS_VUCM_01276] [SWS_VUCM_01277] [SWS_VUCM_01303] [SWS_VUCM_01305] [SWS_VUCM_CONSTR_00003] [SWS_VUCM_CONSTR_00005] [SWS_VUCM_CONSTR_00006] [SWS_VUCM_CONSTR_00009] [SWS_VUCM_CONSTR_00011] [SWS_VUCM_CONSTR_00015] [SWS_VUCM_CONSTR_00018]





Requirement	Description	Satisfied by
[RS_VUCM_00036]	V-UCM shall use platform communication services for interacting with UCMS	[SWS_VUCM_01005] [SWS_VUCM_01015] [SWS_VUCM_01016]
[RS_VUCM_00037]	V-UCM shall ensure it is safe to perform any modification to the vehicle	[SWS_VUCM_00177] [SWS_VUCM_00179] [SWS_VUCM_00183] [SWS_VUCM_00297] [SWS_VUCM_00298] [SWS_VUCM_01109] [SWS_VUCM_01114] [SWS_VUCM_01117] [SWS_VUCM_01138] [SWS_VUCM_01139] [SWS_VUCM_01222] [SWS_VUCM_01228] [SWS_VUCM_01229] [SWS_VUCM_01234] [SWS_VUCM_01240] [SWS_VUCM_01244] [SWS_VUCM_01246] [SWS_VUCM_01275] [SWS_VUCM_01278] [SWS_VUCM_01309] [SWS_VUCM_01310] [SWS_VUCM_CONSTR_00003] [SWS_VUCM_CONSTR_00004] [SWS_VUCM_CONSTR_00005] [SWS_VUCM_CONSTR_00006] [SWS_VUCM_CONSTR_00007] [SWS_VUCM_CONSTR_00009] [SWS_VUCM_CONSTR_00018]
[RS_VUCM_00038]	V-UCM shall interact with driver	[SWS_VUCM_00180] [SWS_VUCM_00182] [SWS_VUCM_01105] [SWS_VUCM_01117] [SWS_VUCM_01118] [SWS_VUCM_01120] [SWS_VUCM_01135] [SWS_VUCM_01222] [SWS_VUCM_01228] [SWS_VUCM_01234] [SWS_VUCM_CONSTR_00017]
[RS_VUCM_00039]	V-UCM shall prevent processing of compromised Vehicle Packages	[SWS_VUCM_00136] [SWS_VUCM_00181] [SWS_VUCM_01221] [SWS_VUCM_01301] [SWS_VUCM_01302] [SWS_VUCM_01306]
[RS_VUCM_00042]	V-UCM shall provide an interface to read the state of an update campaign	[SWS_VUCM_00181] [SWS_VUCM_01017] [SWS_VUCM_01177] [SWS_VUCM_01178] [SWS_VUCM_01203] [SWS_VUCM_01205] [SWS_VUCM_01265] [SWS_VUCM_CONSTR_00016]
[RS_VUCM_00043]	V-UCM shall orchestrate a software update campaign according to the Vehicle Package's Manifest	[SWS_VUCM_00136] [SWS_VUCM_00179] [SWS_VUCM_00180] [SWS_VUCM_00181] [SWS_VUCM_00182] [SWS_VUCM_00210] [SWS_VUCM_01003] [SWS_VUCM_01014] [SWS_VUCM_01015] [SWS_VUCM_01016] [SWS_VUCM_01201] [SWS_VUCM_01207] [SWS_VUCM_01209] [SWS_VUCM_01212] [SWS_VUCM_01228] [SWS_VUCM_01275] [SWS_VUCM_01280] [SWS_VUCM_01301] [SWS_VUCM_01302] [SWS_VUCM_01303] [SWS_VUCM_01305] [SWS_VUCM_01306] [SWS_VUCM_01307] [SWS_VUCM_01308] [SWS_VUCM_CONSTR_00018]
[RS_VUCM_00046]	V-UCM initialization	[SWS_VUCM_01019]

**Table 6.1: Requirements Tracing**

## 7 Functional specification

### 7.1 Functional cluster life-cycle

[SWS\_VUCM\_01205]{DRAFT} **V-UCM internal state persistency** [V-UCM shall persist its state to be able to resume on-going update campaign after an intended or unintended reboot.] ([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00042](#))

### 7.2 Startup

[SWS\_VUCM\_01019]{DRAFT} **V-UCM initialization** [V-UCM shall offer its services only after its internal initialization has been completed, and then report `kRunning` state.] ([RS\\_VUCM\\_00046](#))

This requirement prevents calling V-UCM API while internal initialization is on-going. The concrete initialization tasks are implementation specific.

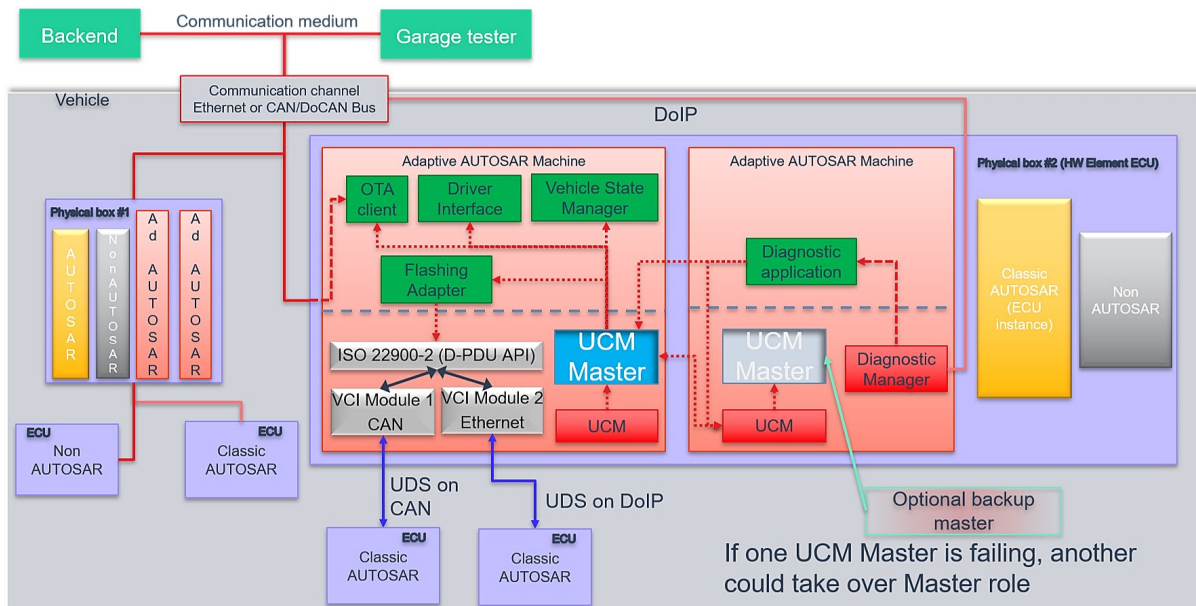
### 7.3 Shutdown

No specific requirements for V-UCM's shutdown

### 7.4 Technical Overview

V-UCM objective is to provide a standard Adaptive Autosar solution to safely and securely update a complete vehicle Over The Air or by a Diagnostic Tester.

V-UCM receives packages from [Backend](#) or Diagnostic tool (through Diagnostic Application), parses and interprets the [Vehicle Package](#), transfers or streams [Software Packages](#) to suitable targets (UCM or Flashing Adapter) and orchestrates the processing, activations and eventual rollbacks. All these actions are what is called a campaign which V-UCM is coordinating. The UCM of the machines in the same network of a V-UCM, candidates target of a campaign, are referred to as [UCMs](#).



**Figure 7.1: Example of v-UCM architecture overview within a vehicle**

The v-UCM could be considered as a set of add-on features that could enrich any UCM instance. Therefore, as per the UCM APIs, the v-UCM APIs are part of the AUTOSAR Adaptive Platform Services. UCM and v-UCM have separate service instances.

The OTA Client establishes a communication between Backend and v-UCM so that they can exchange information of the installed Software Clusters in the vehicle and the Software Clusters available in the Backend. This communication could be triggered by OTA Client with a scheduler and v-UCM to request the updates in case of newly available Software Clusters (pull case) or by Backend to push, for instance, an important security update to a fleet of vehicles (push case). The computation to find new Software Clusters versions and resolution of dependencies between Software Clusters can be either done at v-UCM or Backend.

Vehicle Driver interface Adaptive Application is required if it is needed during an update campaign to interact with vehicle human driver through for instance Human-Machine Interface. Download of packages from a Backend could have various financial costs for the driver depending of communication types, so consent from driver could be suitable.

Vehicle State Manager Adaptive Application is required if it is needed during an update campaign to control the vehicle state for safety purposes. For instance, it could be required for safety to have standing still vehicle, shut-off engine, closed doors, etc. before starting an UCM activation or during its processing.



## 7.5 V-UCM general behavior

The V-UCM acts as a client of the service interface offered by the UCM, already specified in UCM. However, the V-UCM also offers three different service interfaces to OTA Client, Vehicle Driver interface and Vehicle State Manager respectively. V-UCM aggregates UCMs states and can report its status field to a Backend through its OTA Client.

A V-UCM receives a Vehicle Package and transfers or streams Software Package(s) to the UCMs for an AUTOSAR Adaptive Platform Software Cluster update. A Vehicle Package contains instructions for orchestrating updates between ECUs. The V-UCM provides information about ECUs in the vehicle, installed software and update campaign resolution.

**[SWS\_VUCM\_01003] V-UCM checks states of UCMs** [A V-UCM shall check the status of its UCMs are all at `kIdle CurrentStatus` state before starting a campaign.] ([RS\\_VUCM\\_00043](#))

V-UCM should for instance make sure that there is no ongoing diagnostic updates before starting an update campaign by checking the reported state(s) of the UCM(s) to be idle.

**[SWS\_VUCM\_CONSTR\_00018] V-UCM uniqueness in vehicle** [At any given point in time, at most one V-UCM shall provide `VehiclePackageManagement`, `VehicleDriverApplicationInterface` and `VehicleStateManagerInterface` services interfaces per network domain.] ([RS\\_VUCM\\_00043](#), [RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00037](#))

There can be several V-UCMs running in the vehicle as backup but only one at a time can provide services in the vehicle.

## 7.6 UCM identification

For V-UCM to distribute Software Packages to other UCMs, V-UCM has to identify UCMs in vehicle. This identification could be at boot or later but at least before any communication with Backend are engaged. Each UCM has a unique identifier in `Vehicle Package UcmModuleInstantiation` called identifier to help V-UCM transferring packages to targeted UCMs. To get such identifier, V-UCM will perform first a service discovery through `ara::com` to get all UCMs service instances available. Then V-UCM will call `GetId` method for each UCMs returning each corresponding `UcmModuleInstantiation` identifiers.

If an ECU hosting a UCM is replaced physically, it will register its services to the registry at boot up and V-UCM will be able to communicate with UCM(s).

**[SWS\_VUCM\_01005] V-UCM is discovering UCMs in vehicle** [V-UCM shall continuously look for UCM service instances (use of `StartFindService()` call).] ([RS\\_VUCM\\_00036](#))

If a **V-UCM** is failing, another inactive **V-UCM** could be used or activated by **OTA Client**.

Default (at boot) **V-UCM/UCM** hierarchy or priority could be optionally overwritten for each campaign based on **Vehicle Package** content at the condition **OTA Client** could properly parse **Vehicle Packages**.

## 7.7 V-UCM Software Packages transfer or streaming

**V-UCM** has generally same transfer API as **UCM** in order to simplify implementation and reuse code as much as possible (could be shared library between **UCM** and **V-UCM**).

It is necessary to distinguish **Vehicle Package** (**V-UCM** specific) from **Software Packages** transfer.

**[SWS\_VUCM\_01011] TransferVehiclePackage InsufficientMemory** [**TransferVehiclePackage** method shall raise the **ApplicationError kInsufficientMemory** if the **UCM** buffer has not enough resources to process the corresponding **Vehicle Package**.] (*RS\_UCM\_00013*)

**[SWS\_VUCM\_01018]{DRAFT} TransferVehiclePackage kBusyWithCampaign** [**TransferVehiclePackage** method shall return the **ApplicationError kBusyWithCampaign**, if the **OTA Client** wants to start a new campaign, while a campaign is already started and active.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01014] Packages transferring sequence** [**TransferStart** method shall raise the **ApplicationError kUnexpectedPackage** if the **Software Package** name parameter was not a value of the **RequestedPackage** field.] (*RS\_VUCM\_00043*)

**[SWS\_VUCM\_01013] Too big block size received by V-UCM** [In the case the received block size with **TransferData** exceeds the block size returned by **TransferStart** or **TransferVehiclePackage** for the same **TransferId**, **V-UCM** shall raise the **ApplicationError kIncorrectBlockSize**.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01015] Invalid Vehicle Package manifest** [**TransferExit** shall raise the **ApplicationError kInvalidPackageManifest** when a **Vehicle Package** manifest is not compliant with the AUTOSAR schema.] (*RS\_VUCM\_00036, RS\_VUCM\_00043*)

**[SWS\_VUCM\_01016] Invalid Package Manifest** [**V-UCM** shall raise the **ApplicationError kInvalidPackageManifest** in case a manifest file is not compliant with the AUTOSAR schema.] (*RS\_VUCM\_00036, RS\_VUCM\_00043*)

**[SWS\_VUCM\_01017] RequestedPackage field** [The field **RequestedPackage** shall contain the requested **Software Package** name and version as configured in campaign which is modelled by **VehiclePackage**.] (*RS\_VUCM\_00042*)

OTA Client does not know what Software Packages should be transferred in a given campaign contained in a Vehicle Package. OTA Client can know what Software Package is expected to be transferred by subscribing to V-UCM's RequestedPackage field. Version is added to support campaigns which need an update path for a Software Package requiring an intermediate update to a transitional version. In this case the version parameter makes it unambiguous which package version shall be transferred as both have the same name assigned.

## 7.8 V-UCM retry strategy

When V-UCM or OTA Client calls TransferData method and it raises ApplicationError kBlockInconsistent, V-UCM or OTA Client can retry the TransferData method again later for the same block. This behaviour is configured by UcmRetryStrategy.

**[SWS\_VUCM\_01020]{DRAFT} Retry Strategy for BlockInconsistent** [When TransferData returns ApplicationError kBlockInconsistent more than maximumNumberOfRetries within retryIntervalTime, then V-UCM shall cancel the active campaign by transitioning to kCancelling state and delete the failing package.](RS\_VUCM\_00035)

If no retry strategy is needed, the maximum number of attempts can be specified as 0 in UcmRetryStrategy.

When V-UCM calls ProcessSwPackage method and it raises ApplicationError kServiceBusy, V-UCM can retry the ProcessSwPackage method again later for the same package. This behaviour is configured by UcmRetryStrategy.

**[SWS\_VUCM\_00297]{DRAFT} Retry Strategy for ServiceBusy** [When ProcessSwPackage returns ApplicationError kServiceBusy more than maximumNumberOfRetries within retryIntervalTime, then V-UCM shall cancel the active campaign by transitioning to kCancelling state and delete the failing package.](RS\_VUCM\_00035, RS\_VUCM\_00037)

If no retry strategy is needed, the maximum number of attempts can be specified as 0 in UcmRetryStrategy.

When V-UCM calls Activate method and it raises ApplicationError kUpdateSessionRejected, V-UCM can retry the Activate method again later to enter Update Session again. This behaviour is configured by UcmRetryStrategy.

**[SWS\_VUCM\_00298]{DRAFT} Retry Strategy for UpdateSessionRejected** [When Activate returns ApplicationError kUpdateSessionRejected more than maximumNumberOfRetries within retryIntervalTime, then V-UCM shall cancel the active campaign by transitioning to kCancelling state.](RS\_VUCM\_00035, RS\_VUCM\_00037)

If no retry strategy is needed, the maximum number of attempts can be specified as 0 in [UcmRetryStrategy](#).

## 7.9 Adaptive Applications interacting with V-UCM

In order to have interoperability between several vendors platforms, [Adaptive Applications](#) interacting with V-UCM via [ara::com](#) like [OTA Client](#), [Vehicle State Manager](#) or Vehicle Driver Interface have their APIs specified. However, their detailed behaviours are out of scope for this specification document.

### 7.9.1 OTA Client

[OTA Client](#) is an [Adaptive Application](#) that sets communication channel between [Backend](#) and V-UCM. The communication between [Backend](#) and [OTA Client](#) is abstracted and details like protocol are out of scope for this specification document. [OTA Client](#) should make sure [Backend](#) is providing the right information and packages to the vehicle by identifying the vehicle, by for instance sending VIN to [Backend](#).

[OTA Client](#) uses the V-UCM as a service provider via [ara::com](#). Since transferring [Vehicle Packages](#) and [Software Packages](#) from [Backend](#) to V-UCM is [OTA Client](#)'s responsibility, [OTA Client](#) should be able to accommodate any proprietary communication protocol used between [OTA Client](#) and [Backend](#) and convert it into [ara::com](#) transport protocol. [OTA Client](#) should support [UCM Software Packages](#) transfer or streaming as specified in [5], it should then provide at least the following functionality:

- Comply to the requirements of [5] in the context of package transfer between [OTA Client](#) and V-UCM.
- [OTA Client](#) should subscribe to V-UCM's [RequestedPackage](#) field to know what [Software Package](#) is expected to be transferred
- [OTA Client](#) should subscribe to V-UCM's [TransferState](#) field to know what is campaign state
- [OTA Client](#) should subscribe to V-UCM's [VehicleConditionCollection](#) field to eventually make sure vehicle is in a safe state before transferring Packages
- [OTA Client](#) could support multiple data transfers in parallel, as specified in [SWS\_UCM\_00075] of [5]

In addition, [OTA Client](#) could support the ability to pause or resume the package transfer for the current campaign to prioritize the transfer of the packages from a different campaign. The ability of [OTA Client](#) to pause or resume the package transfer might be helpful in the case there is a need to cancel an ongoing campaign at [kTransferring](#) state to allow higher priority campaign to be performed. If a campaign is

cancelled during transfer of packages, `OTA Client` has the possibility to recover the transfers using `GetSwPackages` method (providing `transferId`, `consecutiveBytesReceived` and `consecutiveBlocksReceived`) but could also simply delete the partially transferred packages with `DeleteTransfer` method.

Only one `V-UCM` has to be used by `OTA Clients` per network domain. As `V-UCM` is distributing `Software Packages` and coordinating `UCM` subordinates, `OTA Clients` in the same network domain have to make sure there are no already on-going campaigns when starting a new campaign with `TransferVehiclePackage` method call by checking `V-UCM`'s state with `TransferState` field, in order to avoid any interference and guarantee success of an update campaign.

**[SWS\_VUCM\_01101] Provide information of installed `Software Clusters` in vehicle** [`V-UCM` shall provide a method `GetSwClusterInfo` to return information of all `Software Cluster` that are in state `kPresent`.] (*RS\_VUCM\_00033*)

`V-UCM` can aggregate `Software Cluster` information from several `UCMs` within a vehicle and returns the result to a `Backend` which can compute if there is any new `Software Cluster` available and decide to send to `V-UCM` through `OTA Client` a `Vehicle Package`. It is up to `OTA Client` to make sure the synchronisation of the versions of `Software Packages` present in `Backend` and `Software Clusters` in the vehicles using `GetSwClusterInfo` or `SwPackageInventory` is recent enough before starting a campaign with `TransferVehiclePackage` call.

**[SWS\_VUCM\_01103] Inform `Backend` of needed `Software Packages` for an update** [`On SwPackageInventory` call, `V-UCM` shall compare the supplied list of available `Software Packages` in the `Backend` for the vehicle to its own internal information of present `Software Clusters` in the vehicle and return the list of `Software Packages` selected for update.] (*RS\_VUCM\_00033*)

The `OTA Client` uses this returned `Software Packages` list to request the selected packages to the `Backend`. As required by constraint [SWS\_UCM\_CONSTR\_00014] of [5], each `Software Cluster` corresponds to one `Software Package` and share the same `shortName`.

**[SWS\_VUCM\_CONSTR\_00016]{DRAFT} `OTA Client` use of `RequestedPackage` field** [`When V-UCM` updates the `RequestedPackage` field, the `OTA Client` shall start the transfer of the requested `Software Package`.] (*RS\_VUCM\_00042*)

**[SWS\_VUCM\_01119]{OBSOLETE} Report information of `Software Packages`** [`V-UCM` shall provide a method `GetSwPackages` to return the identifiers, names, versions, Consecutive Bytes Received, Consecutive Blocks Received and states of `Software Packages`.] (*RS\_VUCM\_00035*)

At the invocation of method `GetSwPackages`, `V-UCM` returns the identifiers, names, versions, Consecutive Bytes Received, Consecutive Blocks Received and states of `Software Packages`.

OTA Client can use the `GetSwPackages` method to get the complete picture of the packages that are part of current campaign. Then, based on this information, the OTA Client can either start or continue the transfer of necessary software packages.

## 7.9.2 Vehicle Driver Interface

Vehicle driver interface could be required by legal constraints or communication cost consideration. To support mandatory safety and security critical updates, driver interaction can be used for:

- Requesting transfer, processing or activation permission from vehicle driver
- Notifying vehicle driver of safety and security measures he has to apply to the vehicle in order to proceed to next step into the update campaign

**[SWS\_VUCM\_01105]{OBSOLETE} Interaction of V-UCM with Vehicle Driver** [V-UCM shall provide a method `DriverApproval` in order to receive the confirmation of the vehicle driver's approval.] (*RS\_VUCM\_00038*)

**[SWS\_VUCM\_CONSTR\_00017]{DRAFT} Interaction of V-UCM with Vehicle Driver** [When vehicle driver accepts or denies the campaign, Vehicle Driver Application shall call `DriverApproval` method of `VehicleDriverApplicationInterface` to inform V-UCM of the vehicle driver's decision.] (*RS\_VUCM\_00038*)

The Vehicle Driver Interface `Adaptive Application` could adapt its notification content related to safety by subscribing to the V-UCM's `VehicleConditionCollection` field.

**[SWS\_VUCM\_01117]{DRAFT} V-UCM VehicleConditionCollection field** [V-UCM shall provide to vehicle driver interface the `VehicleConditionCollection` field containing the required safety condition for the campaign as configured in `safetyCondition`.] (*RS\_VUCM\_00038*, *RS\_VUCM\_00037*)

V-UCM can notify vehicle driver with `VehicleConditionCollection` field if the vehicle safety is breached during the update, by for instance popping-up a message.

**[SWS\_VUCM\_01118] V-UCM waiting for vehicle driver approval** [In the case approval from driver is requested as configured in `VehiclePackage`, V-UCM shall wait for `DriverApproval` method with parameter `approval=True` before transitioning state from `kVehiclePackageTransferring` to `kSoftwarePackage_Transferring`, `kSoftwarePackage_Transferring` to `kProcessing` or `kProcessing` to `kActivating`.] (*RS\_VUCM\_00038*)

**[SWS\_VUCM\_CONSTR\_00003] Exclusive use of Vehicle Driver Interface** [Software Integrator shall ensure that only one `Adaptive Application` is using the V-UCM's Vehicle Driver Interface.] (*RS\_VUCM\_00035*, *RS\_VUCM\_00037*)

For example, the integrator may restrict the access of Vehicle Driver Interface from V-UCM by configuring the Identity and Access Management functional cluster accordingly.

**[SWS\_VUCM\_CONSTR\_00004] Unsupported safety by Vehicle driver interface** [In the case the Vehicle Driver Interface *Adaptive Application* does not support at least one safety condition contained in the *VehicleConditionCollection* field, the Vehicle Driver Interface *Adaptive Application* shall call the method *ReportUnsupportedSafetyConditions* with parameter *UnsupportedSafetyConditions* containing all the safety conditions which are not supported by Vehicle Driver Interface *Adaptive Application*.] (*RS\_VUCM\_00037*)

**[SWS\_VUCM\_01120] Provide Software Packages general information** [V-UCM shall provide a method *GetSwPackageDescription* to return the description of each *Software Packages* that are part of current campaign and that are contained in *Vehicle Package*.] (*RS\_VUCM\_00033*, *RS\_VUCM\_00038*)

**[SWS\_VUCM\_01135]{OBSOLETE} Get Software Clusters descriptions from a vehicle** [At *GetSwClusterInfo* method call via *VehicleDriverApplicationInterface* interface, UCM *Master* shall return the descriptions of all *Software Clusters* which are aggregated from all the UCM Subordinates and all Flashing Adapters.] (*RS\_VUCM\_00033*, *RS\_VUCM\_00038*)

### 7.9.3 Vehicle State Manager

*Vehicle State Manager* is collecting states from the several vehicle ECUs and informs V-UCM when the safety state computed based on the safety policy referred in the *Vehicle Package* is changing. If the safety policy is not met, the V-UCM can for instance:

- Inform vehicle driver that the safety conditions are not met to continue the update
- Based on vehicle package configuration, pause the update until policy is met or cancel campaign

**[SWS\_VUCM\_01139]{DRAFT} V-UCM configured behavior when vehicle safety is not met** [When a safety condition is not met, V-UCM shall wait in case *violatedSafetyConditionBehavior* is set to *waitForVehicleSafeState* or cancel the campaign when *violatedSafetyConditionBehavior* is set to *cancelCampaign*.] (*RS\_VUCM\_00037*)

**[SWS\_VUCM\_01109]{DRAFT} V-UCM provides a safety interface** [V-UCM shall set the *VehicleConditionCollection* field taking the value from *safetyCondition* attribute for each *VehicleRolloutStep* of the *VehiclePackage*.] (*RS\_VUCM\_00037*)

*Vehicle State Manager* Adaptive Application can inform any vehicle state changes by calling *PublishSafetyState* method.

**[SWS\_VUCM\_CONSTR\_00005]{DRAFT} Safety state change** [Vehicle State Manager Adaptive Application shall call `PublishSafetyState` method provided by V-UCM when the safety state is changing.](RS\_VUCM\_00035, RS\_VUCM\_00037)

**[SWS\_VUCM\_CONSTR\_00009]{DRAFT} Safety condition change** [Vehicle State Manager Adaptive Application shall call `PublishSafetyState` method provided by V-UCM when at least one state subelement of the field `VehicleConditionCollection` has been changed.](RS\_VUCM\_00035, RS\_VUCM\_00037)

**[SWS\_VUCM\_CONSTR\_00015]{DRAFT} Trigger on kVehicleChecking state** [On transition to `kVehicleChecking` state provided by `CampaignState` from `VehicleStateManagerInterface`, Vehicle State Manager shall first perform checks to assess the post-activation state of the vehicle.](RS\_VUCM\_00035)

Vehicle State Manager could be responsible for performing post-activation checks, interfacing with an application performing such checks, confirming backend is still reachable and further updates are still possible.

**[SWS\_VUCM\_01272]{DRAFT} VehicleCheck call not permitted** [V-UCM shall return `ApplicationError kOperationNotPermitted` if `VehicleCheck` method is called in another V-UCM state than `kVehicleChecking`.](RS\_VUCM\_00035)

**[SWS\_VUCM\_CONSTR\_00006]{DRAFT} Exclusive use of Vehicle State Manager** [System Integrator shall ensure that Vehicle State Manager is the exclusive user of the `PublishSafetyState` method.](RS\_VUCM\_00035, RS\_VUCM\_00037)

For example, the integrator may restrict the access to Vehicle State Manager in configuring the Identity and Access Management functional cluster accordingly.

**[SWS\_VUCM\_01275]{DRAFT} Safety conditions during activation** [V-UCM shall apply the `safetyCondition` of the last `VehicleRolloutStep` to `VehicleConditionCollection` field during the ECUs activations as configured in the `VehiclePackage`.](RS\_VUCM\_00043, RS\_VUCM\_00037)

**[SWS\_VUCM\_CONSTR\_00007]{DRAFT} Unsupported safety conditions by Vehicle State Manager** [In the case the requested `VehicleConditionCollection` field is not referring to an existing safety condition implemented by Vehicle State Manager, the Vehicle State Manager shall call `VehicleStateManagerInterface` service interface `PublishSafetyState` method with parameter `safetyStates` containing at least one value equal to 'NotSupported'.](RS\_VUCM\_00037)

**[SWS\_VUCM\_01278]{DRAFT} v-UCM behaviour in case a safety condition is not supported by Vehicle State Manager** [If there is at least one safety state returned by `PublishSafetyState` method which equals to 'NotSupported', V-UCM shall cancel the campaign and set `UCMMasterResolutionType` to `kUnsupportedSafetyCondition`.](RS\_VUCM\_00037)



### 7.9.4 Flashing Adapter

Flashing Adapter is an application that is used in the case [V-UCM](#) is updating a [AUTOSAR Classic Platform](#) or any platform that can be flashed using diagnostic. It contains OEM specific diagnostic sequences and communicates via `ara::com` with the [V-UCM](#) and the [AUTOSAR Adaptive Platform](#), and uses an implementation of diagnostic protocol data unit application programming interface ([D-PDU API](#)) to communicate with Classic ECUs over the Vehicle Bus.

Flashing Adapter contains the needed logic to put the [AUTOSAR Classic Platform ECUs](#) into the right update mode in order to avoid any timeout issues during update.

The data transfer from Flashing Adapter to the target [ECU](#) via diagnostic communication can be subject to interruptions if communication on a higher priority protocol occurs, e.g. OBD services. In that case the Flashing Adapter can use a project specific strategy to detect the interruption, retry the transfer from the beginning, and decide whether to notify or not the client about the transfer interruption.

**[SWS\_VUCM\_CONSTR\_00011]{DRAFT} Flashing Adapter provided interface**  
[Flashing Adapter shall provide the same `ara::com` service interface as [UCM](#) ([SWS\_VUCM\_00131] of [5]).] ([RS\\_VUCM\\_00035](#))

### 7.9.5 UCM Diagnostic Application

	Diagnostic Application as UCM Client	Diagnostic Application as V-UCM Client
Purpose	Update standalone ECU/Machine without involvement of V-UCM	Update ECU/Machine as part of vehicle update through V-UCM
Flow	Diagnostic Tool -> Diagnostic Manager -> Diagnostic App -> UCM	Diagnostic Tool -> Diagnostic Manager -> Diagnostic App -> V-UCM
Instance	One instance per standalone Adaptive ECU/Machine	One instance per vehicle
Artifacts handled	Receives Software Packages	Receives Vehicle Packages and Software Packages
UCM API (Service)	Package Management	Vehicle Package Management
ECUs/Machines being updated	Adaptive only (incl. Classic Machines on Adaptive ECU, if needed)	Any ECU (Adaptive, Classic, Proprietary)
Implemented by	ECU Vendor and/or OEM	OEM

	Diagnostic Application as UCM Client	Diagnostic Application as V-UCM Client
References	<ul style="list-style-type: none"> <li>• Figures "Architecture overview for diagnostic use case", "Sequence diagram showing the update process", "Sequence diagram showing the data transmission", "Sequence diagram showing the package processing" in the document [5]</li> <li>• Figure "Vehicle Update Architecture" in AUTOSAR_EXP_PlatformDesign [6]</li> <li>• Figure "Interfaces of UCM" in AUTOSAR_EXP_SWArchitecture [3]</li> </ul>	<ul style="list-style-type: none"> <li>• Figure 7.1, Figure 7.2 in this document, Figure "Sequence diagram showing the data transmission" in [5]</li> </ul>

**Table 7.1: The usage of UCM Diagnostic Application**

### 7.9.6 Non Adaptive Platform update

The interface provided by the [AUTOSAR Adaptive Platform](#) in order to update non AUTOSAR Platform complies with the subset of ISO 22900-2:2017 ([D-PDU API](#)) requirements. As this standard's coverage is wide, it is allowed to implement a reduced API that is needed to update for instance a [AUTOSAR Classic Platform](#).

The implementation of the [D-PDU API](#) is processing binary data from the Flashing Adapter and do all of the required session, transport and network layer handling to send and receive the data on the physical vehicle bus with respect to the underlying protocols. The reason of using ISO 22900-2:2017 is to ensure that the specific Flashing Adapter from any vehicle or tool manufacturer can operate on a common software interface and can easily exchange [MVCI](#) (Modular Vehicle Communication Interface) protocol module implementations.

In the case the targeted ECU by an update does not have the capability to switch between current and new [Software Cluster](#), the vehicle package campaign should foresee to download not only the new version but also the currently installed version of the [Software Cluster](#) to be updated in order to make possible a rollback from the new version to the old version of the [Software Cluster](#). The location to store the current [Software Package](#) could be the Flashing Adapter but ultimately it has to be available to Flashing Adapter in order to flash it in case of a rollback.

As an implementation example, the pdu api source code and the communication parameters can be at appendix [F](#).

#### 7.9.6.1 D-PDU API implementation support

**[SWS\_VUCM\_01122]{DRAFT} Supported physical layers by [D-PDU API](#) implementation** [ISO\_11898\_2\_DWCAN (Dual Wire CAN), ISO\_11898\_3\_DWFTCAN (Dual Wire CAN Fault tolerant), SAE\_J2411\_SWCAN (Single Wire CAN) and IEEE\_802\_3(Ethernet) physical layers shall be supported if their respective physical vehicle bus is available inside the ECU] ([RS\\_VUCM\\_00035](#)) All other physical layers present in D-PDU API are optional.

**[SWS\_VUCM\_01123]{DRAFT} Supported application layers by [D-PDU API](#) implementation** [ISO\_15765\_3 (Unified diagnostic services, UDS on CAN, ISO withdrawn UDS), ISO\_14229\_3 (Unified diagnostic services on CAN implementation, UDSON-CAN) and ISO\_14229\_5 (Unified diagnostic services on Internet Protocol implementation, UDSONIP) application layers shall be supported if their respective application layer is available inside the ECU.] ([RS\\_VUCM\\_00035](#))

All other application layers present in [D-PDU API](#) are optional.

**[SWS\_VUCM\_01124]{DRAFT} Supported protocols by [D-PDU API](#) implementation** [ISO UDS on CAN with Application layer ISO\_15765\_3, ISO UDS on CAN with Application layer ISO\_14229\_3 (UDSONCAN) and ISO UDS on DoIP with Application layer ISO\_14229\_5 (UDSONIP) protocols shall be supported.] ([RS\\_VUCM\\_00035](#))

All other protocols are optional.

These protocols are present in 'Table B.2 - Standard protocol combination list' of ISO 22900-2:2017(E).

### 7.9.6.2 Not required D-PDU API concepts

Dynamic Link Libraries for Windows operating system are not required. The Windows installation process out of ISO 22900-2:2017(E) chapter 8.7.2 is not applicable to the [AUTOSAR Adaptive Platform](#) which is using POSIX Operating System.

A [D-PDU API](#) implementation can be split at OSI-Layer 4 into a [D-PDU API](#) implementation on OSI-Layer 5 and the [VCI-Module](#) on OSI-Layers 3 and 4.

The [D-PDU API](#) implementation does not use the [D-PDU API](#) root description file (RDF) as only one [D-PDU API](#) implementation is required for [UCM](#) within an [AUTOSAR Adaptive Platform](#).

The only instance of the [D-PDU API](#) within a [Software Cluster](#) can be statically linked with the Flashing Adapter.

The [D-PDU API](#) implementation does not have to implement a protocol description file.

The supported protocol module types are fixed in the [UCM](#) use case.

The Flashing Adapter can operate the [D-PDU API](#) without using symbolic names and IDs during runtime. If the use case excludes frequent changes to the [MDFs](#), simple Flashing Adapter can even hardcode (e.g. in a header file) all necessary IDs and operate the [D-PDU API](#) without symbolic names.

[D-PDU API](#) implementation does not need to be compatible to SAE J2534-1 and RP 1210a.

The Adaptive Platform does not need any migration path.

[D-PDU API](#) implementation does not need to implement the IOCTL filter data structure.

### 7.9.6.3 Not required D-PDU API functions

[PDU LockResource\(\)](#) and [PDU UnlockResource\(\)](#) are used to lock and unlock exclusive access to a [ComLogicalLink](#) in case of parallel usage of the [D-PDU API](#) implementation by multiple applications on the same physical communication link. Flashing of a Classic ECU always requires some exclusive access and should be handled in the [AUTOSAR Adaptive Platform](#) itself.

The [D-PDU API](#) implementation does not have to implement the parameter [PDU\\_IOCTL\\_RESET](#)

**[SWS\_VUCM\_01131]{DRAFT} PDUIoctl(PDU\_IOCTL\_RESET)** [The call of PDUIoctl(PDU\_IOCTL\_RESET) shall return the error code PDU\_ERR\_ID\_NOT\_SUPPORTED, if PDU\_IOCTL\_RESET is not implemented.] ([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01132]{DRAFT} PDUIoctl(PDU\_IOCTL\_START\_MSG\_FILTER), PDUIoctl(PDU\_IOCTL\_CLEAR\_MSG\_FILTER), PDUIoctl(PDU\_IOCTL\_STOP\_MSG\_FILTER)** [The call of PDUIoctl() with any of the parameters PDU\_IOCTL\_START\_MSG, PDU\_IOCTL\_CLEAR\_MSG\_FILTER, PDU\_IOCTL\_SEND\_BREAK shall return the error code PDU\_ERR\_ID\_NOT\_SUPPORTED.] ([RS\\_VUCM\\_00035](#))

The parameters PDU\_IOCTL\_START\_MSG, PDU\_IOCTL\_CLEAR\_MSG\_FILTER and PDU\_IOCTL\_CLEAR\_MSG\_FILTER are intended for the PassThru-Mode for com-primitives and therefore an implementation is not required for the Flashing Adapter.

The IOCTL command PDU\_IOCTL\_SEND\_BREAK is used to send a break signal on the ComLogicalLink. A break signal can only be sent on certain physical layers (e.g. SAE J1850 VPW physical links and UART physical links) which are not supported by UCM.

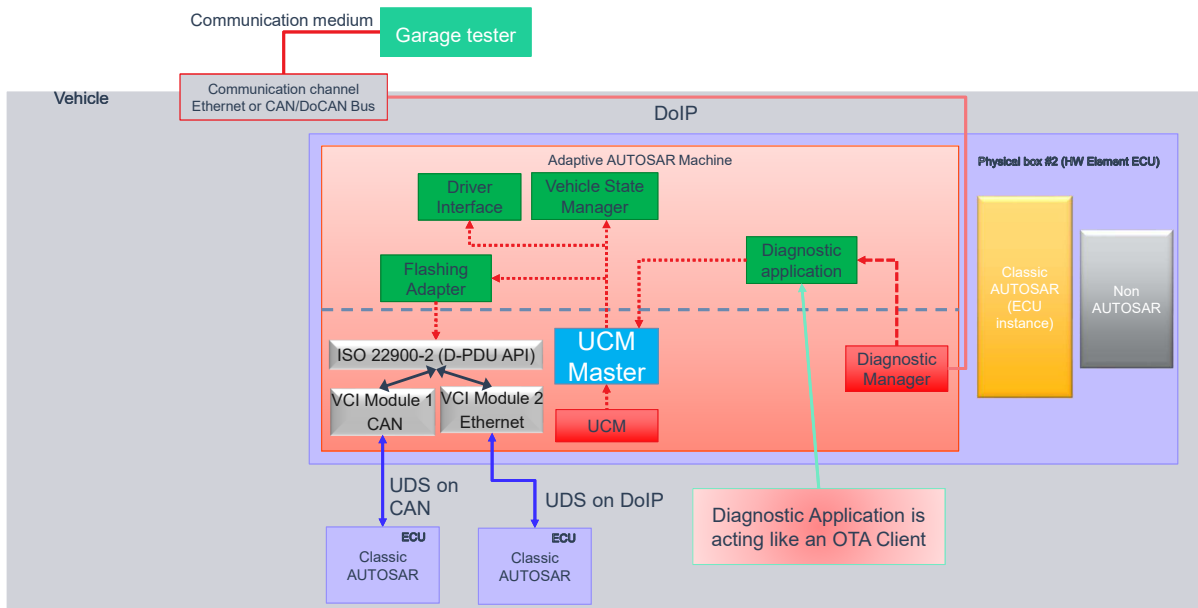
The [D-PDU API](#) implementation of the [AUTOSAR Adaptive Platform](#) does not have to implement the returned codes PDU\_ERR\_CABLE\_UNKNOWN, PDU\_ERR\_RSC\_LOCKED, PDU\_ERR\_RSC\_NOT\_LOCKED, PDU\_ERR\_API\_SW\_OUT\_OF\_DATE and PDU\_ERR\_MODULE\_FW\_OUT\_OF\_DATE.

There is no cable attached to the ECU and therefore no cable detection return code PDU\_ERR\_CABLE\_UNKNOWN could occur.

Locking is not required for the Flashing Adapter, therefore PDU\_ERR\_RSC\_LOCKED and PDU\_ERR\_RSC\_NOT\_LOCKED return code could not occur.

There is no separation of [D-PDU API-Software](#) with the [MVCI](#) protocol module firmware required in the [AUTOSAR Adaptive Platform](#), so PDU\_ERR\_API\_SW\_OUT\_OF\_DATE and PDU\_ERR\_MODULE\_FW\_OUT\_OF\_DATE return codes could not occur.

**7.9.6.4 Classic platform update with V-UCM and diagnostic tool**



**Figure 7.2: Classic platform update with V-UCM and diagnostic tool**

The Diagnostic Manager connects the Diagnostic tool to the Adaptive Platform. The diagnostic application is acting like an *OTA Client* and uses the *V-UCM* services to push *Vehicle Packages* and *Software Packages*.

Note that this approach allows to update through Diagnostic Tool not only Classic ECU, but also Adaptive or Proprietary/Legacy Machines.

## 7.10 Status reporting

V-UCM supports a mechanism to provide the state of an update campaign typically to OTA Client, Vehicle Driver Application and Vehicle State Manager.

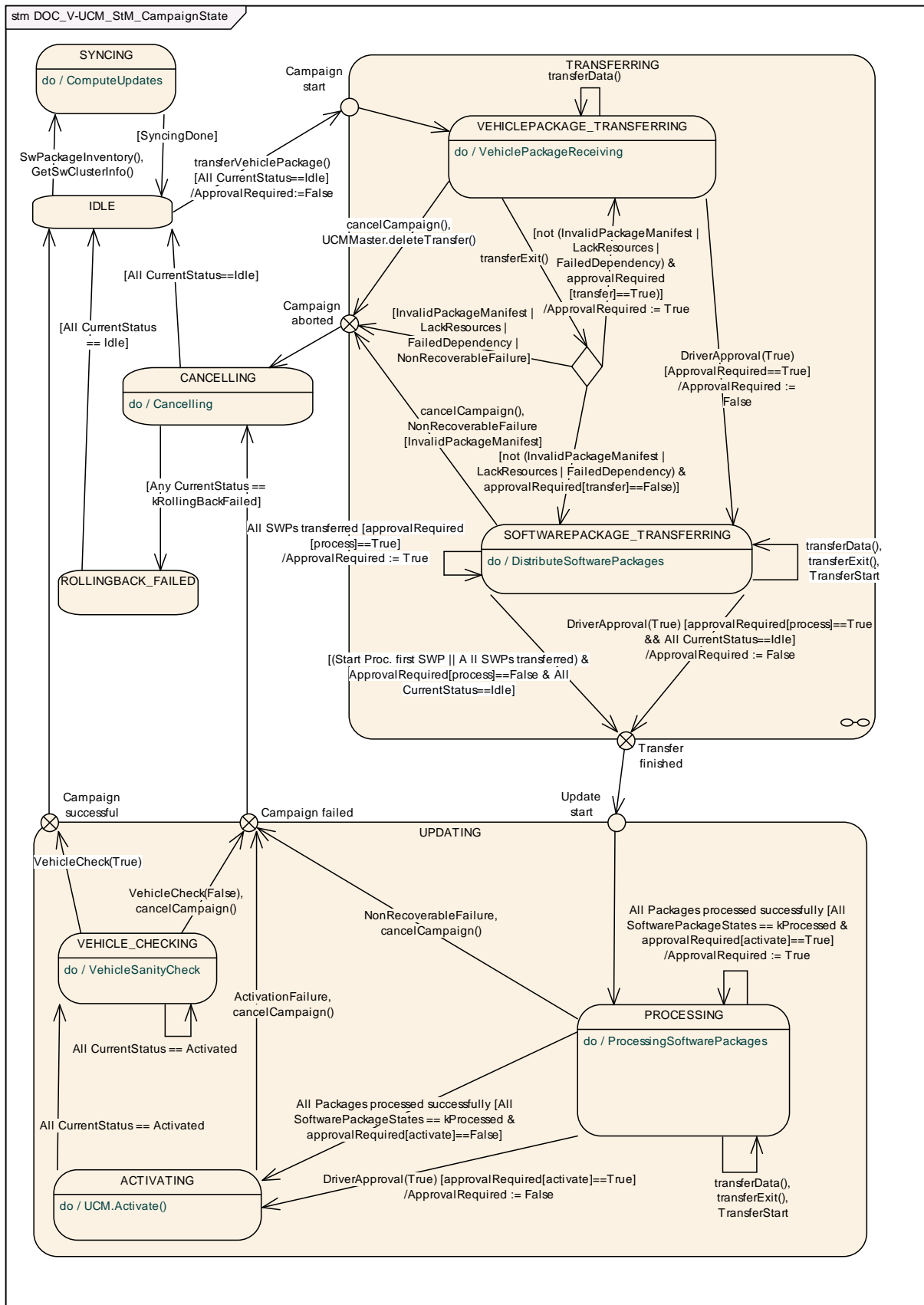
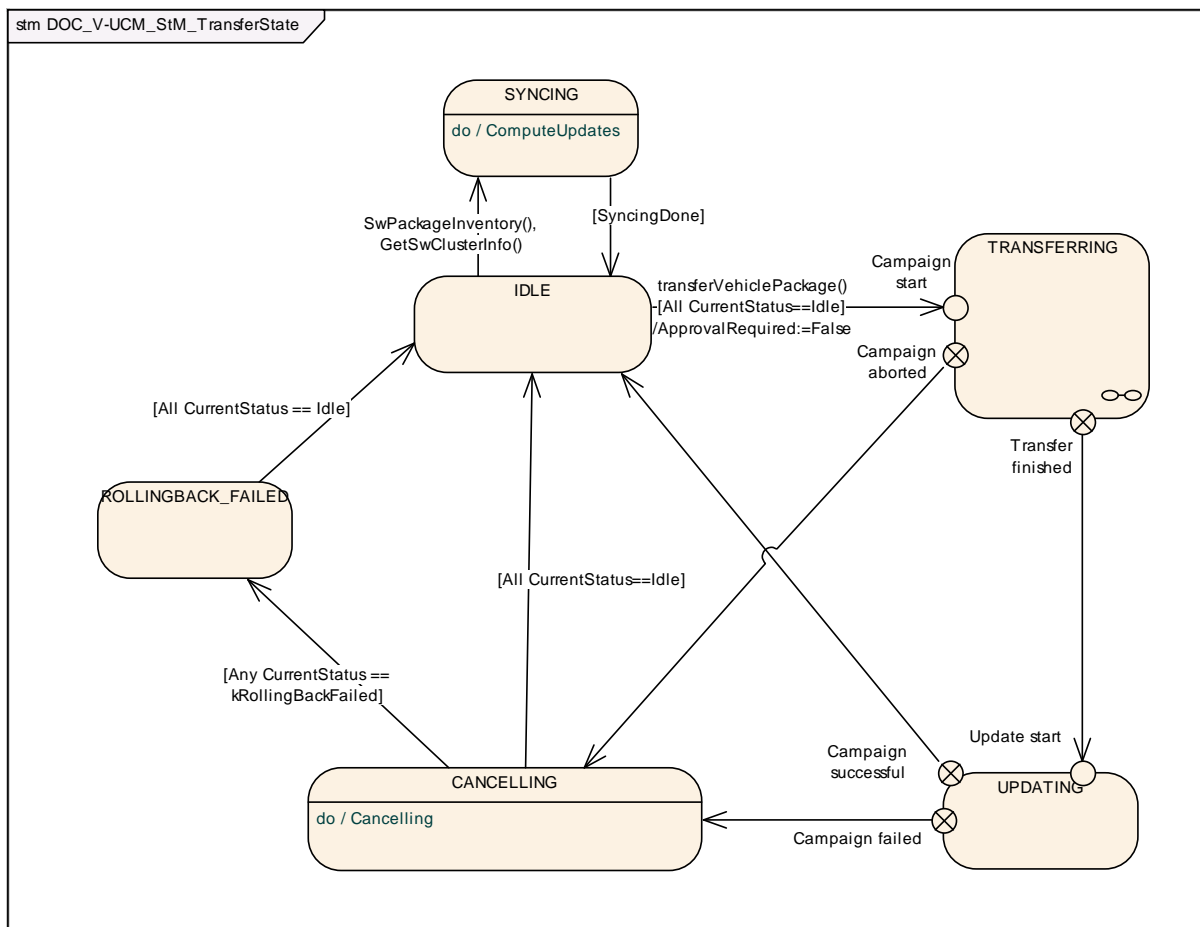


Figure 7.3: Campaign State Machine (CampaignState field)





**Figure 7.4: Campaign State Machine for OTA Client (*TransferState* field)**

Diagrams 7.3 and 7.4 do not include behaviour after reset ([SWS\_VUCM\_01205] for more details)

**[SWS\_VUCM\_01201] Sequential orchestration of campaigns** [V-UCM shall orchestrate at most a single campaign at any one time.](RS\_VUCM\_00043)

**[SWS\_VUCM\_01265] TransferState field** [V-UCM shall provide the state of a campaign over the *TransferState* field of the V-UCM’s *VehiclePackageManagement* service interface.](RS\_VUCM\_00042)

**[SWS\_VUCM\_01203] CampaignState field** [V-UCM shall provide the state of a campaign over the *CampaignState* field of the V-UCM *VehicleDriverApplication-Interface* and *VehicleStateManagerInterface*.](RS\_VUCM\_00042) There is an overview of the campaign state machine in Fig. 7.3 detailing V-UCM campaign states and transitions.

**7.10.1 States**

**[SWS\_VUCM\_01204] Initial state** [V-UCM shall have *kIdle* default state.](RS\_VUCM\_00035)

**[SWS\_VUCM\_01207] Trigger on `kSoftwarePackage_Transferring` state** [On transition to `kSoftwarePackage_Transferring` state and if all UCMS part of the campaign are in `kIdle` state, V-UCM shall start or resume transferring (`TransferStart` and `TransferData` as well as `TransferExit` if no streaming required) the software packages to the UCMS according to the campaign orchestration.]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00043](#))

**[SWS\_VUCM\_01209] Trigger on `kProcessing` state** [On transition to `kProcessing` state, V-UCM shall call `ProcessSwPackage` method to UCMS to start or resume processing the software packages ready for processing according to the campaign orchestration.]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00043](#))

**[SWS\_VUCM\_00210] Transferring of software packages on `kProcessing` state** [If V-UCM is in `kProcessing` state, V-UCM shall transfer `Software Packages` to the UCMS according to the campaign orchestration.]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00043](#))

**[SWS\_VUCM\_01212] Trigger on `kActivating` state** [On transition to `kActivating` state, V-UCM shall ask UCMS to activate the software with `Activate` method call according to the campaign orchestration as configured in the `VehiclePackage`.]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00043](#))

**[SWS\_VUCM\_01214]{DRAFT} Final action on `kVehicleChecking` state** [If V-UCM is in `kVehicleChecking` state and receives the method `VehicleCheck` call with parameter `vehicleCheckResolution=True`, V-UCM shall secondly commit (`Finish`) the software on all UCMS part of the campaign.]([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01215]{DRAFT} Trigger on `kCancelling` state** [On transition to `kCancelling` state, V-UCM shall rollback (`Rollback`) the activated or being verified `Software Clusters`, and revert the processed packages (`RevertProcessedSwPackages`) of the UCMS part of the campaign.]([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01216] Final action on `kCancelling` state** [If V-UCM is in `kCancelling` state and the rollback of software on all UCMS is successful (successful `Rollback` and transition from `kRollingBack` to `kRolledBack`), V-UCM shall secondly commit (`Finish`) the software on all UCMS part of the campaign.]([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01217] Monitoring of UCMS** [V-UCM shall subscribe to the `CurrentStatus` field, in order to follow the current campaign from the state of the UCMS.]([RS\\_VUCM\\_00035](#))

## 7.10.2 States Transitions

**[SWS\_VUCM\_01218] Transition from `kIdle` state to `kSyncing` state** [If V-UCM is in `kIdle` state for `CampaignState` field and `kIdle` state for `TransferState` field, V-UCM shall enter the `kSyncing` state for `CampaignState` and `kSyncing` for

TransferState on a request to GetSwClusterInfo or SwPackageInventory.] (RS\_VUCM\_00035, RS\_VUCM\_00033)

**[SWS\_VUCM\_01219] Transition from kSyncing state to kIdle state** [If V-UCM is in kSyncing state for CampaignState field and kSyncing state for TransferState field, V-UCM shall enter the kIdle state and kIdle for TransferState on completion of GetSwClusterInfo or SwPackageInventory.] (RS\_VUCM\_00035)

**[SWS\_VUCM\_01220] Transition from kIdle state to kVehiclePackageTransferring and kTransferring states** [If V-UCM is in kIdle state for CampaignState field and kIdle state for TransferState field, V-UCM shall enter the kVehiclePackageTransferring state and kTransferring state on successful completion of TransferVehiclePackage.] (RS\_VUCM\_00035)

**[SWS\_VUCM\_01221] Transition from kVehiclePackageTransferring state and kTransferring state to kCancelling state** [If V-UCM is in kVehiclePackageTransferring state for CampaignState field and kTransferring state for TransferState field, V-UCM shall enter the kCancelling state for CampaignState and kCancelling state for TransferState on unsuccessful completion of TransferExit (Vehicle Package) or successful completion of DeleteTransfer (Vehicle Package) or non recoverable error of TransferData.] (RS\_VUCM\_00035, RS\_VUCM\_00039)

**[SWS\_VUCM\_01222] Transition from kVehiclePackageTransferring state to kSoftwarePackage\_Transferring state** [If V-UCM is in kVehiclePackageTransferring state, V-UCM shall enter the kSoftwarePackage\_Transferring state on successful completion of TransferExit (Vehicle Package).] (RS\_VUCM\_00035, RS\_VUCM\_00037, RS\_VUCM\_00038)

**[SWS\_VUCM\_01227] Transition from kSoftwarePackage\_Transferring state and kTransferring state to kCancelling state** [If V-UCM is in kSoftwarePackage\_Transferring state for CampaignState and kTransferring state for TransferState, V-UCM shall enter the kCancelling state for CampaignState and kCancelling state for TransferState on successful cancellation request (CancelCampaign) or if there is a non recoverable transfer failure from one of the UCMS.] (RS\_VUCM\_00035)

**[SWS\_VUCM\_01228] Transition from kSoftwarePackage\_Transferring state and kTransferring state to kProcessing state and kUpdating state** [When V-UCM is in kSoftwarePackage\_Transferring state for CampaignState and kTransferring state for TransferState, if all Software Packages are ready for processing, all Software Packages from all UCMS are at state kTransferred or at least one Software Package started being processed by ProcessSwPackage call to one UCM according to the campaign orchestration, V-UCM shall enter the kProcessing state for CampaignState and kUpdating state for TransferState.] (RS\_VUCM\_00035, RS\_VUCM\_00037, RS\_VUCM\_00038, RS\_VUCM\_00043)

**[SWS\_VUCM\_01229]{DRAFT} SafetyConditions while processing stream** [In the case there is transition from `kSoftwarePackage_Transferring` state to `kProcessing` state, the `SafetyConditions` for `kProcessing` state shall apply even though there are `Software Packages` transferring.]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00037](#))

It is integrator's responsibility to make sure in this use case that safety conditions for Processing will also cover safety approach of transferring.

**[SWS\_VUCM\_01234] Transition from `kProcessing` state to `kActivating` state** [If `V-UCM` is in `kProcessing` state and all software packages of the campaign have been successfully (successful `ProcessSwPackage`) processed and all `UCMs` part to the campaign are in the `kReady` state, `V-UCM` shall enter the `kActivating` state.]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00037](#), [RS\\_VUCM\\_00038](#))

**[SWS\_VUCM\_01236] Transition from `kProcessing` state and `kUpdating` state to `kCancelling` state** [If `V-UCM` is in `kProcessing` state for `CampaignState` and `kUpdating` state for `TransferState`, `V-UCM` shall enter the `kCancelling` state for `CampaignState` and `kCancelling` state for `TransferState` on successful cancellation request (`CancelCampaign`) or in case of non recoverable processing failure of one of the `UCMs`.]([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01239] Transition from `kActivating` state and `kUpdating` state to `kCancelling` state** [If `V-UCM` is in `kActivating` state for `CampaignState` and `kUpdating` state for `TransferState`, `V-UCM` shall enter the `kCancelling` state for `CampaignState` and `kCancelling` state for `TransferState` if any `UCMs` part of the campaign unsuccessfully (unsuccessful `Activate` and transition from `kVerifying` to `kRollingBack`) completed activation.]([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01240] Transition from `kActivating` state to `kVehicleChecking` state** [If `V-UCM` is in `kActivating` state, `V-UCM` shall enter the `kVehicleChecking` state if all `UCMs` part of the campaign successfully (successful `Activate` and transition from `kVerifying` to `kActivated`) completed activation.]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00037](#))

**[SWS\_VUCM\_01241]{DRAFT} Transition from `kVehicleChecking` state and `kUpdating` state to `kCancelling` state** [If `V-UCM` is in `kVehicleChecking` state for `CampaignState` and `kUpdating` state for `TransferState`, and receives the method `VehicleCheck` call with parameter `vehicleCheckResolution=False`, `V-UCM` shall enter the `kCancelling` state for `CampaignState` and `kCancelling` state for `TransferState`.]([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01242] Transition from `kVehicleChecking` state and `kUpdating` state to `kIdle` state** [If `V-UCM` is in `kVehicleChecking` state for `CampaignState` and `kUpdating` state for `TransferState` and all `UCMs` part of the campaign transitioned from `kCleaningUp` to `kIdle`, `V-UCM` shall enter the `kIdle` state for `CampaignState` and `kIdle` state for `TransferState`.]([RS\\_VUCM\\_00035](#))

**[SWS\_VUCM\_01276]{DRAFT} Transition from `kRollingBackFailed` state to `kIdle` state** [If `V-UCM` is in `kRollingBackFailed` state for `CampaignState` field and

`kRollingBackFailed` state for `TransferState` field, `V-UCM` shall enter the `kIdle` state and `kIdle` for `TransferState` when all `UCM CurrentStatus` fields are at `kIdle`.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01277]{DRAFT} Transition from `kCancelling` state to `kRollingBackFailed` state** [If `V-UCM` is in `kCancelling` state for `CampaignState` field and `kCancelling` state for `TransferState` field, `V-UCM` shall enter the `kRollingBackFailed` state and `kRollingBackFailed` for `TransferState` when at least one `UCM CurrentStatus` field transitions to `kRollingBackFailed`.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01243] Transition from `kCancelling` state to `kIdle` state** [If `V-UCM` is in `kCancelling` state for `CampaignState` and `kCancelling` state for `TransferState` and all `UCMs` part of the campaign transitioned from `kCleaningUp` to `kIdle`, `V-UCM` shall enter the `kIdle` state for `CampaignState` and `kIdle` state for `TransferState`.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01246]{DRAFT} Unreachable UCM during update campaign** [In case a `UCM` is not reachable by `V-UCM` during an update campaign (from `kTransferring` or `kUpdating`), `V-UCM` shall transit to `kCancelling` state for `CampaignState` and `kCancelling` state for `TransferState`.] (*RS\_VUCM\_00035*, *RS\_VUCM\_00037*)

## 7.11 Campaign cancelling

`CancelCampaign` method could be used at garage to unlock a blocked update. Details on action by `V-UCM`, like cleaning up the several `UCMs`, changing `AUTOSAR Adaptive Platform` states, etc. are implementation specific.

In case an update campaign was cancelled, a new update campaign could use again the already transferred `Software Packages`. `V-UCM` could list transferred `Software Packages` by calling the `UCMs` with `GetSwPackages`.

**[SWS\_VUCM\_01244]{DRAFT} Cancellation of an update campaign shall be possible** [Method `CancelCampaign` from `V-UCM` shall trigger a campaign cancel from `kTransferring` or `kUpdating` states (from `TransferState` field).] (*RS\_VUCM\_00035*, *RS\_VUCM\_00037*)

**[SWS\_VUCM\_01270]{DRAFT} New campaign disabling** [`V-UCM` shall remain in `kIdle` when a `CancelCampaign` method has been called with `disableCampaign` parameter set.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01271]{DRAFT} New campaign enabling** [Method `AllowCampaign` from `V-UCM` shall reallow new campaign after a `CancelCampaign` method was called with `disableCampaign` parameter set.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01280]{DRAFT} Maximum campaign duration** [`V-UCM` shall cancel the campaign when the campaign duration (time from when `V-UCM` transitions to `kTransfer`, considering only the active or powered Machine time and excluding time

waiting for driver approvals) exceeds `maximumDurationOfCampaign`] (*RS\_VUCM\_00043*)

Time counter starts after `TransferVehiclePackage` method call and have to be persisted in order to resume after shutdown and restart of Machine.

**[SWS\_VUCM\_01273]{DRAFT} CancelCampaign kCancelFailed error** [`CancelCampaign` shall raise the error `ApplicationError kCancelFailed` in case cancelling of a campaign fails.] (*RS\_VUCM\_00035*)

**[SWS\_VUCM\_01274]{DRAFT} CancelCampaign kOperationNotPermitted error** [`CancelCampaign` shall raise the error `ApplicationError kOperationNotPermitted` in case the V-UCM states are at `kIdle`, `kSyncing` or `kCancelling`.] (*RS\_VUCM\_00035*)

## 7.12 Campaign Reporting

After campaign is finished (finish method has been sent to all UCMs), V-UCM should report to `Backend` server status of the vehicle, with for instance updated information of `Software Clusters` present in vehicle.

**[SWS\_VUCM\_01247] Method to read History Report** [V-UCM shall provide a method `GetCampaignHistory` to retrieve all actions that have been performed by V-UCM when exiting state `kUpdating` from a specific time window.] (*RS\_VUCM\_00034*)

**[SWS\_VUCM\_01248] Content of History Report** [V-UCM shall save the point in time when `TransferVehiclePackage` method is called and the point in time when `kIdle` state is entered from any state except `kSyncing` based on `timeBaseResource` and the campaign result in `CampaignHistoryType`] (*RS\_VUCM\_00034*)

The elements of `UCMHistoryVectorType` [SWS\_VUCM\_00254] are not sorted.

**[SWS\_VUCM\_01279] Keep history of Driver notification during campaign** [If all `approvalRequired` values are false in the campaign when V-UCM is recording campaign history, V-UCM shall unset `driverNotified` attribute otherwise set it.] (*RS\_VUCM\_00034*)

**[SWS\_VUCM\_01266]{DRAFT} Subordinate Not Available On The Network** [V-UCM shall record persistently into history the error `kSubordinateNotAvailableOnTheNetwork` in case one of the UCM subordinate involved in the current campaign stops offering its `Service Interface`.] (*RS\_VUCM\_00034*)

**[SWS\_VUCM\_01267]{DRAFT} Vehicle State Manager Communication Error** [V-UCM shall record persistently the error `kVehicleStateManagerCommunicationError` into history in case the communication with `Vehicle State Manager` is not possible.] (*RS\_VUCM\_00034*)

[SWS\_VUCM\_01268]{DRAFT} **Vehicle Driver Interface Communication Error** [V-UCM shall record persistently the error `kVehicleDriverInterfaceCommunicationError` into history in case the communication with Vehicle Driver Interface is no longer possible.] (RS\_VUCM\_00034)

[SWS\_VUCM\_01269]{DRAFT} **Campaign cancellation history** [If `CancelCampaign` method is called, V-UCM shall record persistently the error `kCampaignCancelled` into history.] (RS\_VUCM\_00034)

## 7.13 Content of Vehicle Package



Figure 7.5: Vehicle package overview

A `Vehicle Package` is typically assembled by an `OEM Backend`. A `Vehicle Package` has to be modelled as a so-called `VehiclePackage` which describes the content of the `Vehicle Package`. It contains a collection of `Software Package Manifests` extracted from `Backend` packages stored in the `Backend` database. These `Software Packages` have to be modelled as a so-called `SoftwarePackage` which describes the content of the `Software Package`. A `Vehicle Package` contains only one `Vehicle Package Manifest`. Several archive file formats could be used for the `Vehicle Package` or `Software Package` like .zip, gz, bz2, etc. `UCM` and `V-UCM` could implement several extraction algorithm compatible with several archiving formats and identify the right format to apply by analysing the file header.

It is possible that within an update campaign, several `Machine` or `ECUs` need to be updated/installed/removed by groups. Some `Software Clusters` could require re-boot of `Machine` or `ECU`, some just a restart of `Adaptive Application` or nothing (waiting passively for next reboot) to get activated. To optimize a campaign or fulfil dependencies, it could be required to activate `Software Clusters` one after the other or several at once. To support all possible campaigns, the `Vehicle Package` includes a model describing this coordination. It also contains a way to identify the several involved `UCMs` for packages distribution within the vehicle and potentially overwriting default `V-UCM` for this specific campaign.

You can find below for information purpose a description of the information that must be contained in Vehicle Package manifest:

- Repository: uri, repository or diagnostic address, for history, tracking and security purposes
- Vehicle description: vehicle description
- Vehicle Driver notifications: it might be needed to ask vehicle driver if V-UCM can start transferring *Software Packages*, processing it and activating it but also inform him of the necessary safety requirements if applicable.
- Safety policy: safety policy index to be used as argument to subscribe a field to vehicle safety manager. With this field, V-UCM will be informed at any time of campaign if vehicle safety is met or not.
- V-UCM identifiers list: defines backup V-UCMs
- Campaign orchestration: You can refer to [7] for more details. This campaign model allows to group activation of several UCMs and group *Software Packages* processing and transferring.

**[SWS\_VUCM\_01301]{DRAFT} Vehicle Package authentication** [The *Vehicle Package* shall be successfully authenticated by the V-UCM using *CryptoServiceCertificate* at *TransferData* or *TransferExit* call before any transfer of the *Software Packages*, otherwise raise *ApplicationError kOperationNotPermitted*.] (*RS\_VUCM\_00039*, *RS\_VUCM\_00043*)

If *Vehicle Package* is not yet authenticated, V-UCM does not accept any transfer of *Software Packages* that are not confirmed to be part of Campaign for security purpose.

**[SWS\_VUCM\_01302]{DRAFT} Vehicle Package authentication failure** [In case *Vehicle Package* authentication fails at *TransferExit* call, V-UCM shall raise the *ApplicationError kAuthenticationFailed*.] (*RS\_VUCM\_00039*, *RS\_VUCM\_00043*)

**[SWS\_VUCM\_01303]{DRAFT} Dependencies between Software Clusters** [V-UCM shall check dependencies based on *Vehicle Package Manifest*, *Software Packages Manifests* and information on already installed *Software Clusters* and their dependencies retrieved from UCMs using Package Manager service interface method *GetSwClusterManifestInfo* before a transfer of *Software Packages*.] (*RS\_VUCM\_00035*, *RS\_VUCM\_00043*)

The vehicle dependency tree could be resolved by the backend after calling *GetSwClusterInfo*.

**[SWS\_VUCM\_01311] Semantic versioning** [V-UCM shall compute *SoftwareCluster* dependency check comparing only *MajorVersion* and *MinorVersion*.] (*RS\_VUCM\_00033*)



For `SoftwareCluster` dependency check [SWS\_UCM\_00319] or `Vehicle Package` compatibility against V-UCM [SWS\_VUCM\_01308], `PatchVersion` and additional labels of `StrongRevisionLabelString` are not considered.

The `Vehicle Package` contains a `Vehicle Package` manifest and `Software Packages` manifests of ARXML format in order to have interoperability between vendors.

**[SWS\_VUCM\_01305]{DRAFT} `Vehicle Package` format** [When the V-UCM receives a `Vehicle Package` whose `Vehicle Package` manifest and `Software Package` manifests formats are not ARXML, then the V-UCM shall return `ApplicationError kInvalidPackageManifest` from either `TransferData` or `TransferExit` method call and transition from `kVehiclePackageTransferring` or `kTransferring` to `kIdle`.] (*RS\_VUCM\_00035, RS\_VUCM\_00043*)

**[SWS\_VUCM\_01307]{DRAFT} `Vehicle Package` format not supported** [V-UCM shall return `ApplicationError kUnsupportedPackageFormat` from `TransferData` or `TransferExit` methods call in the case the `Vehicle Package` format is not supported.] (*RS\_VUCM\_00043*)

**[SWS\_VUCM\_01306] `TransferExit` Invalid package manifest** [`TransferExit` shall raise the error `ApplicationError kInvalidPackageManifest` upon reception of an invalid manifest.] (*RS\_VUCM\_00039, RS\_VUCM\_00043*)

**[SWS\_VUCM\_01308]{DRAFT} Check `Vehicle Package` version compatibility against V-UCM version** [At `TransferExit` call, V-UCM shall raise `ApplicationError kIncompatiblePackageVersion` if the version for the `Vehicle Package` transferred expressed by `minimumSupportedUcmMasterVersion` attribute is higher than the current version of V-UCM expressed by `version` attribute.] (*RS\_VUCM\_00043*)

## 7.14 Vehicle update security and confidentiality

The methods `GetSwClusterInfo`, `SwPackageInventory` and `GetHistory` could use private or confidential information.

**[SWS\_VUCM\_CONSTR\_00013]{DRAFT} Confidential information protection** [The `VehiclePackageManagement` and `VehicleDriverApplicationInterface` interfaces shall only be called over secure communication channel providing confidentiality protection.] (*RS\_VUCM\_00033*)

The `GetSwClusterInfo`, `SwPackageInventory`, `GetCampaignHistory` and `GetSwPackages` methods are using data that could identify vehicle user and therefore should be protected for confidentiality.

## 8 API specification

There are no APIs defined in this release.

## 9 Service Interfaces

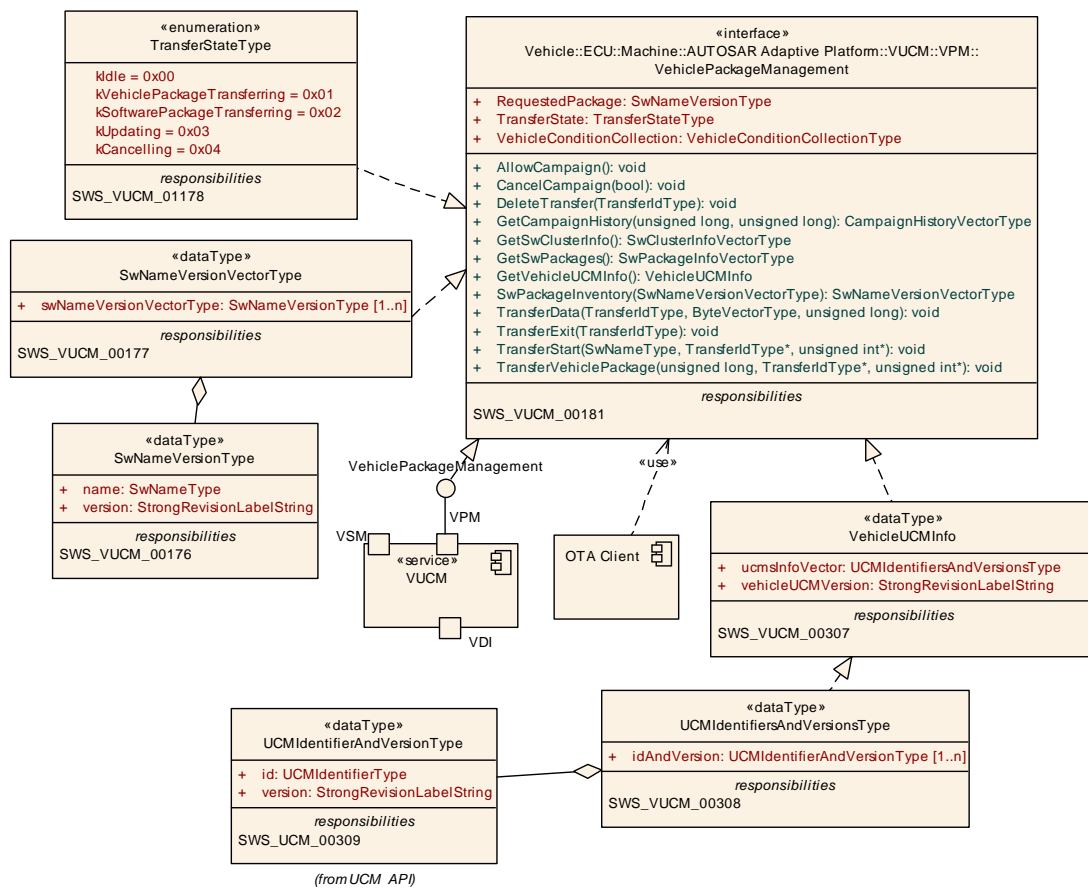
This chapter lists all provided and required service interfaces of the **V-UCM**.

Tables are generated out of 'arxml' folder content.

### 9.1 Type definitions

This chapter lists all types provided by the **V-UCM**.

The following figures are informative and only meant to support reader having global view of V-UCM types and service interfaces.



**Figure 9.1: V-UCM Vehicle Package Management Service Interface**

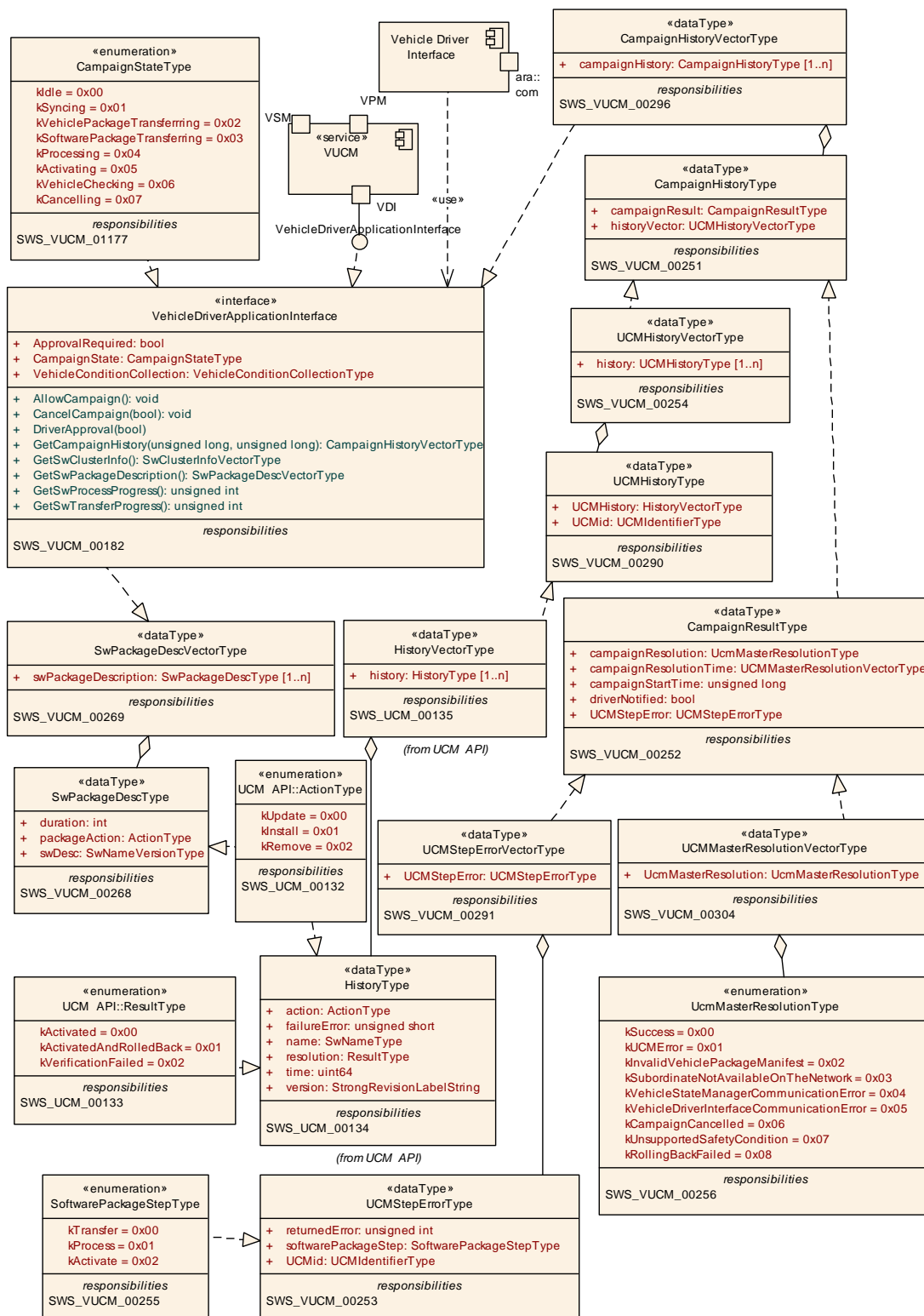
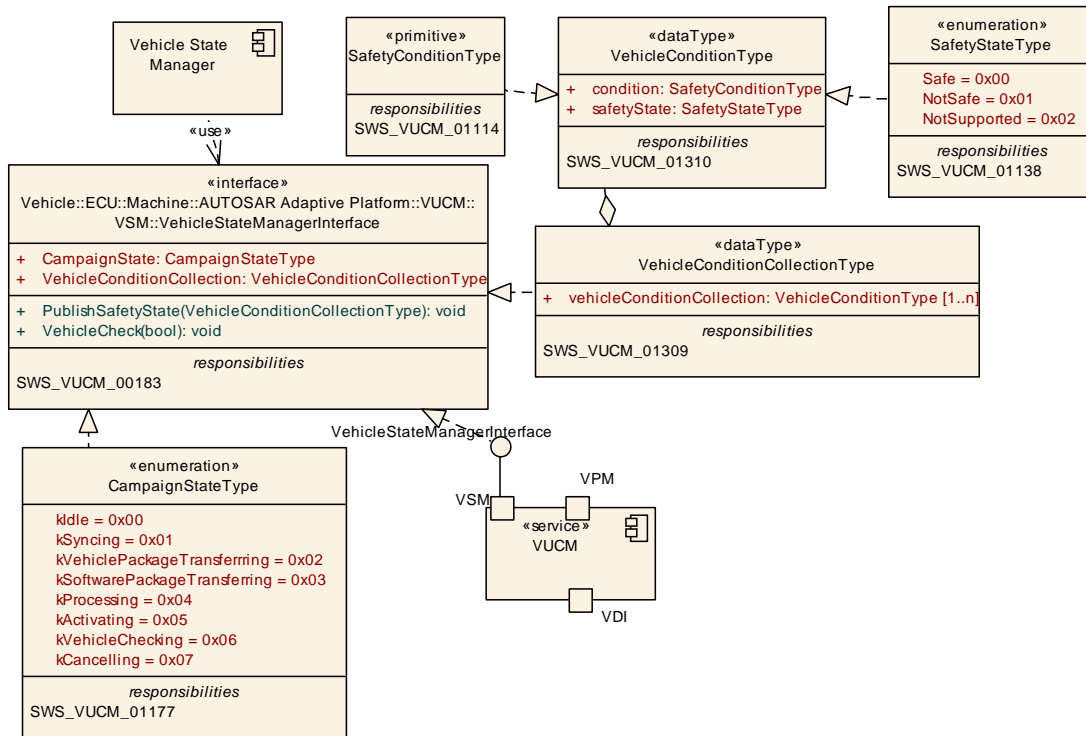


Figure 9.2: V-UCM Vehicle Driver Application Interface



**Figure 9.3: V-UCM Vehicle State Manager Interface**

### 9.1.1 CampaignHistoryType

#### [SWS\_VUCM\_00251]{DRAFT} Definition of ImplementationDataType Campaign HistoryType

<b>Name</b>	CampaignHistoryType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	STRUCTURE
<b>Sub-elements</b>	campaignResult CampaignResultType historyVector UCMHistoryVectorType repository UriString
<b>Derived from</b>	-
<b>Description</b>	Campaign history

|(RS\_VUCM\_00034)

### 9.1.2 CampaignHistoryVectorType

[SWS\_VUCM\_00296]{DRAFT} Definition of ImplementationDataType CampaignHistoryVectorType [

<b>Name</b>	CampaignHistoryVectorType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	VECTOR <CampaignHistoryType>
<b>Derived from</b>	-
<b>Description</b>	Represents a list of Campaign history

](RS\_VUCM\_00034)

### 9.1.3 CampaignResultType

[SWS\_VUCM\_00252]{DRAFT} Definition of ImplementationDataType CampaignResultType [

<b>Name</b>	CampaignResultType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	STRUCTURE
<b>Sub-elements</b>	campaignResolution UCMMasterResolutionVectorType UCMStepError UCMStepErrorVectorType campaignStartTime uint64_t campaignResolutionTime uint64_t driverNotified bool
<b>Derived from</b>	-
<b>Description</b>	Campaign Resolution. UCM gets time from Time Sync Functional Cluster via UcmToTime BaseResourceMapping.timeBaseResource

](RS\_VUCM\_00034)

### 9.1.4 UCMMasterResolutionVectorType

[SWS\_VUCM\_00304]{DRAFT} Definition of ImplementationDataType UCMMasterResolutionVectorType [

<b>Name</b>	UCMMasterResolutionVectorType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	VECTOR <UCMMasterResolutionType>
<b>Derived from</b>	-
<b>Description</b>	Vector of V-UCM errors

](RS\_VUCM\_00034)

### 9.1.5 UCMMasterResolutionType

#### [SWS\_VUCM\_00256]{DRAFT} Definition of ImplementationDataType UCMMasterResolutionType [

<b>Name</b>	UCMMasterResolutionType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	TYPE_REFERENCE	
<b>Derived from</b>	uint8_t	
<b>Description</b>	Campaign resolution	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
kSuccess	0x00	Successful campaign
kUCMError	0x01	UCM error
kInvalidVehiclePackageManifest	0x02	Vehicle Package manifest is invalid
kSubordinateNotAvailableOnTheNetwork	0x03	UCM subordinate not reachable
kVehicleStateManagerCommunicationError	0x04	Communication error with Vehicle State Manager
kVehicleDriverInterfaceCommunicationError	0x05	Communication error with Vehicle Driver Interface
kCampaignCancelled	0x06	Campaign was cancelled
kUnsupportedSafetyCondition	0x07	Safety condition not supported by Vehicle Driver Interface Adaptive Application
kRollingBackFailed	0x08	One UCM failed to revert changes introduced with processed packages.

](RS\_VUCM\_00034)

### 9.1.6 UCMStepErrorVectorType

#### [SWS\_VUCM\_00291]{DRAFT} Definition of ImplementationDataType UCMStepErrorVectorType [

<b>Name</b>	UCMStepErrorVectorType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	VECTOR <UCMStepErrorType>	
<b>Derived from</b>	-	
<b>Description</b>	Vector of UCM's errors	

](RS\_VUCM\_00034)

### 9.1.7 UCMStepErrorType

[SWS\_VUCM\_00253]{DRAFT} Definition of ImplementationDataType UCMStepErrorType [

<b>Name</b>	UCMStepErrorType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	STRUCTURE
<b>Sub-elements</b>	ucmId UCMIdentifierType softwarePackageStep SoftwarePackageStepType returnedError uint8_t
<b>Derived from</b>	-
<b>Description</b>	UCM Error

](RS\_VUCM\_00034)

### 9.1.8 SoftwarePackageStepType

[SWS\_VUCM\_00255]{DRAFT} Definition of ImplementationDataType SoftwarePackageStepType [

<b>Name</b>	SoftwarePackageStepType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	TYPE_REFERENCE	
<b>Derived from</b>	uint8_t	
<b>Description</b>	UCM Software Package step at which error occurred	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
kTransfer	0x00	Software Package transfer
kProcess	0x01	Software Package processing
kActivate	0x02	Software Cluster activation

](RS\_VUCM\_00034)

### 9.1.9 UCMHistoryType

[SWS\_VUCM\_00290]{DRAFT} Definition of ImplementationDataType UCMHistoryType [

<b>Name</b>	UCMHistoryType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	STRUCTURE
<b>Sub-elements</b>	ucmId UCMIdentifierType historyVector HistoryVectorType
<b>Derived from</b>	-
<b>Description</b>	History of an UCM

](RS\_VUCM\_00034)



### 9.1.10 UCMHistoryVectorType

#### [SWS\_VUCM\_00254]{DRAFT} Definition of ImplementationDataType UCMHistoryVectorType [

<b>Name</b>	UCMHistoryVectorType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	VECTOR <UCMHistoryType>
<b>Derived from</b>	-
<b>Description</b>	Vector of UCM sub histories

]([RS\\_VUCM\\_00034](#))

### 9.1.11 CampaignStateType

#### [SWS\_VUCM\_01177]{DRAFT} Definition of ImplementationDataType CampaignStateType [

<b>Name</b>	CampaignStateType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	TYPE_REFERENCE	
<b>Derived from</b>	uint8_t	
<b>Description</b>	Represents the status of Campaign.	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
kIdle	0x00	V-UCM is ready to start a software update campaign.
kSyncing	0x01	V-UCM is providing the list of installed SWCLs (GetSwClusterInfo) or computing the list of SWCLs to install (SwPackageInventory).
kVehiclePackageTransferring	0x02	A vehicle package is being transferred to V-UCM.
kSoftwarePackage_ Transferring	0x03	V-UCM is transferring software packages to the UCM subordinates.
kProcessing	0x04	The processing of software packages on UCM subordinates is ongoing. The transferring of software packages may still occur.
kActivating	0x05	The activation of SWCLs on UCM subordinates is ongoing.
kVehicleChecking	0x06	V-UCM is performing post-activation checks (OEM specific).
kCancelling	0x07	V-UCM is rolling-back the activated SWCLs on the UCM subordinates.
kRollingBackFailed	0x08	One UCM failed to revert changes introduced with processed packages.

]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00042](#))

### 9.1.12 TransferStateType

#### [SWS\_VUCM\_01178]{DRAFT} Definition of ImplementationDataType Transfer StateType [

<b>Name</b>	TransferStateType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	TYPE_REFERENCE	
<b>Derived from</b>	uint8_t	
<b>Description</b>	Represents the state of an update from OTA Client perspective.	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
kIdle	0x00	V-UCM is ready to start a software update campaign.
kSyncing	0x01	V-UCM is providing the list of installed SWCLs (GetSwClusterInfo) or computing the list of SWCLs to install (SwPackageInventory).
kTransferring	0x02	Vehicle or Software Packages are being transferred.
kUpdating	0x03	Software Clusters are being updated in the vehicle.
kCancelling	0x04	An error occurred, campaign is being cancelled, reverting changes.
kRollingBackFailed	0x05	One UCM failed to revert changes introduced with processed packages.

] ([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00042](#))

### 9.1.13 SafetyConditionType

#### [SWS\_VUCM\_01114]{DRAFT} Definition of ImplementationDataType SafetyConditionType [

<b>Name</b>	SafetyConditionType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	STRING	
<b>Derived from</b>	-	
<b>Description</b>	The type of the Safety Conditions.	

] ([RS\\_VUCM\\_00037](#))

### 9.1.14 VehicleConditionCollectionType

#### [SWS\_VUCM\_01309]{DRAFT} Definition of ImplementationDataType VehicleConditionCollectionType [

<b>Name</b>	VehicleConditionCollectionType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	VECTOR < <a href="#">VehicleConditionType</a> >	
<b>Derived from</b>	-	
<b>Description</b>	Represents a dynamic size array of vehicle safety conditions.	

] ([RS\\_VUCM\\_00037](#))

### 9.1.15 VehicleConditionType

[SWS\_VUCM\_01310]{DRAFT} Definition of ImplementationDataType VehicleConditionType [

<b>Name</b>	VehicleConditionType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	STRUCTURE
<b>Sub-elements</b>	condition <a href="#">SafetyConditionType</a> state <a href="#">SafetyStateType</a>
<b>Derived from</b>	-
<b>Description</b>	Safety state of one safety condition.

]([RS\\_VUCM\\_00037](#))

### 9.1.16 SafetyStateType

[SWS\_VUCM\_01138]{DRAFT} Definition of ImplementationDataType SafetyStateType [

<b>Name</b>	SafetyStateType	
<b>Namespace</b>	ara::vucm	
<b>Kind</b>	TYPE_REFERENCE	
<b>Derived from</b>	uint8_t	
<b>Description</b>	Represents the vehicle safety state.	
<b>Range / Symbol</b>	<b>Limit</b>	<b>Description</b>
Safe	0x00	Safe Safety State.
NotSafe	0x01	Not safe Safety State.
NotSupported	0x02	Unsupported Safety State.

]([RS\\_VUCM\\_00037](#))

### 9.1.17 SwNameVersionVectorType

[SWS\_VUCM\_00177]{DRAFT} Definition of ImplementationDataType SwNameVersionVectorType [

<b>Name</b>	SwNameVersionVectorType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	VECTOR <SwNameVersionType>
<b>Derived from</b>	-
<b>Description</b>	Represents a dynamic size array of Software Name and Version

]([RS\\_VUCM\\_00037](#))

### 9.1.18 VehicleUCMInfo

#### [SWS\_VUCM\_00307]{DRAFT} Definition of ImplementationDataType VehicleUCMInfo [

<b>Name</b>	VehicleUCMInfo
<b>Namespace</b>	ara::vucm
<b>Kind</b>	STRUCTURE
<b>Sub-elements</b>	vehicleUCMVersion StrongRevisionLabelString ucmsInfoVector UCMIdentifiersAndVersionsType
<b>Derived from</b>	-
<b>Description</b>	Represents version information of Vehicle UCM and a dynamic size array of UCM subordinates Identifier and Version.

](RS\_VUCM\_00035)

### 9.1.19 UCMIdentifiersAndVersionsType

#### [SWS\_VUCM\_00308]{DRAFT} Definition of ImplementationDataType UCMIdentifiersAndVersionsType [

<b>Name</b>	UCMIdentifiersAndVersionsType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	VECTOR <UCMIdentifierAndVersionType>
<b>Derived from</b>	-
<b>Description</b>	Represents a vector of UCM Module Instantiation numbers and versions of UCMs.

](RS\_VUCM\_00035)

### 9.1.20 SwPackageDescType

#### [SWS\_VUCM\_00268]{DRAFT} Definition of ImplementationDataType SwPackageDescType [

<b>Name</b>	SwPackageDescType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	STRUCTURE
<b>Sub-elements</b>	swDesc SwNameVersionType packageAction ActionType duration uint32_t
<b>Derived from</b>	-
<b>Description</b>	Contains general information related to SoftwarePackage that can be used by Human Interface.

](RS\_VUCM\_00033)

### 9.1.21 SwPackageDescVectorType

#### [SWS\_VUCM\_00269]{DRAFT} Definition of ImplementationDataType SwPackageDescVectorType [

<b>Name</b>	SwPackageDescVectorType
<b>Namespace</b>	ara::vucm
<b>Kind</b>	VECTOR <SwPackageDescType>
<b>Derived from</b>	-
<b>Description</b>	Represents a dynamic size array of SwPackageDescType.

](RS\_VUCM\_00033)

## 9.2 Provided Service Interfaces

### 9.2.1 Vehicle Package Management

This chapter lists all provided service interfaces of the V-UCM to OTA Client Adaptive Application.

Port

#### [SWS\_VUCM\_00178]{DRAFT} Definition of Port VehiclePackageManagement provided by functional cluster VUCM [

<b>Name</b>	VehiclePackageManagement		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	VehiclePackageManagement
<b>Description</b>	Provide services like receiving and dispatching packages, provide history and status of update campaigns, generally used by OTA Client.		
<b>Variation</b>			

](RS\_VUCM\_00035)

Service Interface

#### [SWS\_VUCM\_00181]{DRAFT} Definition of ServiceInterface VehiclePackageManagement [

<b>Name</b>	VehiclePackageManagement
<b>Namespace</b>	ara::vucm

<b>Field</b>	TransferState
<b>Description</b>	The current status of Campaign from an OTA Client perspective.
<b>Type</b>	TransferStateType
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Field</b>	RequestedPackage
<b>Description</b>	Software Package to be transferred to V-UCM, containing Software Package name and version as defined in Vehicle Package.
<b>Type</b>	SwNameVersionType
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Field</b>	VehicleConditionCollection
<b>Description</b>	A set of safety conditions along with their corresponding states.
<b>Type</b>	VehicleConditionCollectionType
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Method</b>	CancelCampaign								
<b>Description</b>	This method aborts an ongoing campaign processing of a Vehicle Package.								
<b>FireAndForget</b>	false								
<b>Parameter</b>	disableCampaign								
	<table border="1"> <tr> <td><b>Description</b></td> <td>To forbid new campaign</td> </tr> <tr> <td><b>Type</b></td> <td>bool</td> </tr> <tr> <td><b>Variation</b></td> <td></td> </tr> <tr> <td><b>Direction</b></td> <td>IN</td> </tr> </table>	<b>Description</b>	To forbid new campaign	<b>Type</b>	bool	<b>Variation</b>		<b>Direction</b>	IN
<b>Description</b>	To forbid new campaign								
<b>Type</b>	bool								
<b>Variation</b>									
<b>Direction</b>	IN								
<b>Application Errors</b>	<table border="1"> <tr> <td>kOperationNotPermitted</td> <td>The operation is not supported in the current context.</td> </tr> </table>	kOperationNotPermitted	The operation is not supported in the current context.						
kOperationNotPermitted	The operation is not supported in the current context.								
<b>Application Errors</b>	<table border="1"> <tr> <td>kCancelFailed</td> <td>Cancel failed.</td> </tr> </table>	kCancelFailed	Cancel failed.						
kCancelFailed	Cancel failed.								

<b>Method</b>	AllowCampaign
<b>Description</b>	To allow a new campaign to start
<b>FireAndForget</b>	false

<b>Method</b>	DeleteTransfer								
<b>Description</b>	Delete a transferred Software or Vehicle Package.								
<b>FireAndForget</b>	false								
<b>Parameter</b>	id								
	<table border="1"> <tr> <td><b>Description</b></td> <td>Transfer ID of the currently running request.</td> </tr> <tr> <td><b>Type</b></td> <td>TransferIdType</td> </tr> <tr> <td><b>Variation</b></td> <td></td> </tr> <tr> <td><b>Direction</b></td> <td>IN</td> </tr> </table>	<b>Description</b>	Transfer ID of the currently running request.	<b>Type</b>	TransferIdType	<b>Variation</b>		<b>Direction</b>	IN
<b>Description</b>	Transfer ID of the currently running request.								
<b>Type</b>	TransferIdType								
<b>Variation</b>									
<b>Direction</b>	IN								
<b>Application Errors</b>	<table border="1"> <tr> <td>kInvalidTransferId</td> <td>The Transfer ID is invalid.</td> </tr> </table>	kInvalidTransferId	The Transfer ID is invalid.						
kInvalidTransferId	The Transfer ID is invalid.								
<b>Application Errors</b>	<table border="1"> <tr> <td>kOperationNotPermitted</td> <td>The operation is not supported in the current context.</td> </tr> </table>	kOperationNotPermitted	The operation is not supported in the current context.						
kOperationNotPermitted	The operation is not supported in the current context.								

<b>Method</b>	GetCampaignHistory	
<b>Description</b>	Getter method to retrieve all actions that have been performed by V-UCM.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	timestampGE	
	<b>Description</b>	Earliest timestamp (inclusive)
	<b>Type</b>	uint64_t
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Parameter</b>	timestampLT	
	<b>Description</b>	Latest timestamp (exclusive)
	<b>Type</b>	uint64_t
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Parameter</b>	campaignHistory	
	<b>Description</b>	The history of all actions that have been performed by V-UCM.
	<b>Type</b>	CampaignHistoryVectorType
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetSwClusterInfo	
<b>Description</b>	This method returns the information of the Software Clusters present in the Adaptive Platform, aggregated from UCM Subordinates or Flashing Adapters.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	swInfo	
	<b>Description</b>	List of installed SoftwareClusters that are in state kPresent.
	<b>Type</b>	SwClusterInfoVectorType
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetSwPackages	
<b>Description</b>	This method returns the Software Packages that are part of current campaign handled by V-UCM.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	packages	
	<b>Description</b>	List of Software Packages.
	<b>Type</b>	SwPackageInfoVectorType
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetVehicleUCMInfo	
<b>Description</b>	This method returns Vehicle UCM version and list of UCM Identifiers and Versions.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	vehicleUCMInfo	
	<b>Description</b>	version of available Vehicle UCM and UCM subordinates.
	<b>Type</b>	VehicleUCMInfo
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	SwPackageInventory	
<b>Description</b>		
<b>FireAndForget</b>	false	
<b>Parameter</b>	availableSoftwarePackages	
	<b>Description</b>	List of available Software Packages in Backend corresponding to VIN.
	<b>Type</b>	SwNameVersionVectorType
	<b>Variation</b>	
<b>Parameter</b>	requiredSoftwarePackages	
	<b>Description</b>	List of Software Packages to be sent to V-UCM.
	<b>Type</b>	SwNameVersionVectorType
	<b>Variation</b>	
<b>Direction</b>	IN	
<b>Direction</b>	OUT	

<b>Method</b>	TransferData	
<b>Description</b>	Block-wise transfer of a Software or Vehicle Package to V-UCM.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	id	
	<b>Description</b>	Transfer ID.
	<b>Type</b>	TransferIdType
	<b>Variation</b>	
<b>Parameter</b>	data	
	<b>Description</b>	Data block of the Software or Vehicle Package.
	<b>Type</b>	ByteVectorType
	<b>Variation</b>	
<b>Parameter</b>	blockCounter	
	<b>Description</b>	Block counter value of the current block.
	<b>Type</b>	uint64_t
	<b>Variation</b>	
<b>Direction</b>	IN	
<b>Application Errors</b>	kOperationNotPermitted	The operation is not supported in the current context.
<b>Application Errors</b>	kInvalidTransferId	The Transfer ID is invalid.
<b>Application Errors</b>	kIncorrectBlock	The same block number is received twice.
<b>Application Errors</b>	kIncorrectBlockSize	The size of the block exceeds the provided block size from TransferStart or TransferVehiclePackage.
<b>Application Errors</b>	kIncorrectSize	The size of the Software or Vehicle Package exceeds the provided size in TransferStart.
<b>Application Errors</b>	kInsufficientMemory	Insufficient memory to perform operation.
<b>Application Errors</b>	kTransferFailed	UCM cannot persist transferred block.
<b>Application Errors</b>	kBlockInconsistent	Consistency check for transferred block failed.







<b>Application Errors</b>	<code>kUnsupportedPackageFormat</code>	The Vehicle Package or Software Package archiving format is not supported.
<b>Application Errors</b>	<code>kAuthenticationFailed</code>	Package authentication failed.
<b>Application Errors</b>	<code>kInvalidPackageManifest</code>	Package manifest could not be read.
<b>Application Errors</b>	<code>kIncompatiblePackageVersion</code>	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or V-UCM.
<b>Application Errors</b>	<code>kPackageInconsistent</code>	Package integrity check failed.
<b>Application Errors</b>	<code>kOldVersion</code>	Software Package version is too old.

<b>Method</b>	TransferExit	
<b>Description</b>	Finish the transfer of a Software or Vehicle Package to V-UCM.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	id	
	<b>Description</b>	Transfer ID of the currently running request.
	<b>Type</b>	TransferIdType
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Application Errors</b>	<code>kOperationNotPermitted</code>	The operation is not supported in the current context.
<b>Application Errors</b>	<code>kInvalidTransferId</code>	The Transfer ID is invalid.
<b>Application Errors</b>	<code>kInsufficientData</code>	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
<b>Application Errors</b>	<code>kAuthenticationFailed</code>	Package authentication failed.
<b>Application Errors</b>	<code>kUnsupportedPackageFormat</code>	The Vehicle Package or Software Package archiving format is not supported.
<b>Application Errors</b>	<code>kPackageInconsistent</code>	Package integrity check failed.
<b>Application Errors</b>	<code>kIncompatiblePackageVersion</code>	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or V-UCM.
<b>Application Errors</b>	<code>kInvalidPackageManifest</code>	Package manifest could not be read.
<b>Application Errors</b>	<code>kMissingDependencies</code>	Activation is not allowed because dependencies are missing.
<b>Application Errors</b>	<code>kOldVersion</code>	Software Package version is too old.

<b>Method</b>	TransferStart	
<b>Description</b>	Start the transfer of a Software Package. The name of the Software Package to be transferred to V-UCM must be provided. V-UCM will generate a Transfer ID for subsequent calls to TransferData, TransferExit, DeleteTransfer. Size of Software Package to be used to transfer to UCM subordinate is available in the Vehicle Package and its contained Software Package Manifests.	
<b>FireAndForget</b>	false	





<b>Parameter</b>	softwarePackageName	
	<b>Description</b>	Software Package Short Name of the Software Package to be transferred.
	<b>Type</b>	SwNameType
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Parameter</b>	id	
	<b>Description</b>	Return TransferId.
	<b>Type</b>	TransferIdType
	<b>Variation</b>	
	<b>Direction</b>	OUT
<b>Parameter</b>	blockSize	
	<b>Description</b>	Size of the blocks to be received with TransferData method.
	<b>Type</b>	uint32_t
	<b>Variation</b>	
	<b>Direction</b>	OUT
<b>Application Errors</b>	<a href="#">kUnexpectedPackage</a>	The Software Package name does not correspond to the RequestedPackage field value.
<b>Application Errors</b>	<a href="#">kInsufficientMemory</a>	Insufficient memory to perform operation.

<b>Method</b>	TransferVehiclePackage	
<b>Description</b>	Start the transfer of a Vehicle Package. The size of the Vehicle Package to be transferred to V-UCM must be provided. V-UCM will generate a Transfer ID for subsequent calls to TransferData, TransferExit, ProcessSwPackage, DeleteTransfer. This call starts a new campaign.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	size	
	<b>Description</b>	Size (in bytes) of the Vehicle Package to be transferred.
	<b>Type</b>	uint64_t
	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Parameter</b>	id	
	<b>Description</b>	Return TransferId.
	<b>Type</b>	TransferIdType
	<b>Variation</b>	
	<b>Direction</b>	OUT
<b>Parameter</b>	blockSize	
	<b>Description</b>	Size of the blocks to be received with TransferData method.
	<b>Type</b>	uint32_t
	<b>Variation</b>	
	<b>Direction</b>	OUT
<b>Application Errors</b>	<a href="#">kBusyWithCampaign</a>	Campaign has already started.
<b>Application Errors</b>	<a href="#">kNewCampaignDisabled</a>	New campaigns are disabled, calling AllowCampaign will enable new campaigns.
<b>Application Errors</b>	<a href="#">kInsufficientMemory</a>	Insufficient memory to perform operation.

|(RS\_VUCM\_00033, RS\_VUCM\_00034, RS\_VUCM\_00035, RS\_VUCM\_00039, RS\_VUCM\_00042, RS\_VUCM\_00043)

## 9.2.2 Vehicle Driver Application Interface

This chapter lists all provided service interfaces of the [V-UCM](#) to the Vehicle Driver Adaptive Application.

Port

### [SWS\_VUCM\_00180]{DRAFT} Definition of Port VehicleDriverApplicationInterface provided by functional cluster VUCM [

<b>Name</b>	VehicleDriverApplicationInterface		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	<a href="#">VehicleDriverApplicationInterface</a>
<b>Description</b>	To be used by Vehicle Driver Interface Adaptive Application in order to interact with human user.		
<b>Variation</b>			

]([RS\\_VUCM\\_00038](#), [RS\\_VUCM\\_00043](#))

Service Interface

### [SWS\_VUCM\_00182]{DRAFT} Definition of ServiceInterface VehicleDriverApplicationInterface [

<b>Name</b>	VehicleDriverApplicationInterface
<b>Namespace</b>	ara::vucm

<b>Field</b>	ApprovalRequired
<b>Description</b>	Flag to inform Adaptive Application if approval from Vehicle Driver is required at current state based on Vehicle Package Manifest.
<b>Type</b>	bool
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Field</b>	CampaignState
<b>Description</b>	The current status of Campaign.
<b>Type</b>	<a href="#">CampaignStateType</a>
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Field</b>	VehicleConditionCollection
<b>Description</b>	A set of safety conditions along with its corresponding states.
<b>Type</b>	<a href="#">VehicleConditionCollectionType</a>
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Method</b>	CancelCampaign	
<b>Description</b>	This method aborts an ongoing campaign processing of a Vehicle Package.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	disableCampaign	
	<b>Description</b>	To forbid new campaign
	<b>Type</b>	bool
	<b>Variation</b>	
<b>Direction</b>	IN	
<b>Application Errors</b>	kOperationNotPermitted	The operation is not supported in the current context.
<b>Application Errors</b>	kCancelFailed	Cancel failed.

<b>Method</b>	AllowCampaign	
<b>Description</b>	To allow a new campaign to start	
<b>FireAndForget</b>	false	

<b>Method</b>	DriverApproval	
<b>Description</b>	Called by Adaptive Application to inform V-UCM of the driver's notification resolution (approve or reject)	
<b>FireAndForget</b>	false	
<b>Parameter</b>	approval	
	<b>Description</b>	Driver's approval or rejection
	<b>Type</b>	bool
	<b>Variation</b>	
<b>Direction</b>	IN	

<b>Method</b>	ReportUnsupportedSafetyConditions	
<b>Description</b>	Called by Adaptive Application to inform V-UCM on all unsupported safety conditions	
<b>FireAndForget</b>	false	
<b>Parameter</b>	UnsupportedSafetyConditions	
	<b>Description</b>	The list of all unsupported safety conditions
	<b>Type</b>	VehicleConditionCollectionType
	<b>Variation</b>	
<b>Direction</b>	IN	

<b>Method</b>	GetCampaignHistory	
<b>Description</b>	Getter method to retrieve all actions that have been performed by V-UCM.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	timestampGE	
	<b>Description</b>	Earliest timestamp (inclusive)
	<b>Type</b>	uint64_t
	<b>Variation</b>	
<b>Direction</b>	IN	
<b>Parameter</b>	timestampLT	
	<b>Description</b>	Latest timestamp (exclusive)
	<b>Type</b>	uint64_t





	<b>Variation</b>	
	<b>Direction</b>	IN
<b>Parameter</b>	history	
	<b>Description</b>	The history of all actions that have been performed by V-UCM.
	<b>Type</b>	<a href="#">CampaignHistoryVectorType</a>
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetSwClusterInfo	
<b>Description</b>	This method returns the information of the Software Clusters present in the Adaptive Platform, aggregated from UCM Subordinates or Flashing Adapters.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	swInfo	
	<b>Description</b>	List of installed SoftwareClusters that are in state kPresent.
	<b>Type</b>	<a href="#">SwClusterInfoVectorType</a>
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetSwPackageDescription	
<b>Description</b>	This method returns the general information of the Software Packages that are part of current campaign handled by V-UCM.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	packages	
	<b>Description</b>	List of Software Packages.
	<b>Type</b>	<a href="#">SwPackageDescVectorType</a>
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetSwProcessProgress	
<b>Description</b>	Get the progress (0 - 100%) of the ProcessSwPackage calls executed by UCM subordinates for every Software Packages of the campaign.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	progress	
	<b>Description</b>	The progress of the ProcessSwPackage calls executed by UCM subordinates for every Software Packages of the campaign (0% - 100%). 0x00 ... 0x64, 0xFF for ""No information available""
	<b>Type</b>	uint8_t
	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetSwTransferToUCMProgress	
<b>Description</b>	Get the aggregated progress (0 - 100%) of the Software Packages being transferred to UCM Subordinates.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	progress	
	<b>Description</b>	The progress of the Software Packages being transferred to UCM Subordinates (0% - 100%). 0x00 ... 0x64, 0xFF for ""No information available""
	<b>Type</b>	uint8_t





	<b>Variation</b>	
	<b>Direction</b>	OUT

<b>Method</b>	GetSwTransferToUCMMasterProgress	
<b>Description</b>	Get the progress (0 - 100%) of the Software Packages being transferred to V-UCM.	
<b>FireAndForget</b>	false	
<b>Parameter</b>	progress	
	<b>Description</b>	The progress of the Software Packages being transferred to V-UCM (0% - 100%). 0x00 ... 0x64, 0xFF for "No information available"
	<b>Type</b>	uint8_t
	<b>Variation</b>	
	<b>Direction</b>	OUT

|(RS\_VUCM\_00033, RS\_VUCM\_00034, RS\_VUCM\_00038, RS\_VUCM\_00043)

### 9.2.3 Vehicle State Manager

This chapter lists all provided service interfaces of the V-UCM to the Vehicle State Manager Adaptive Application.

Port

**[SWS\_VUCM\_00179]{DRAFT} Definition of Port VehicleStateManagerInterface provided by functional cluster VUCM** [

<b>Name</b>	VehicleStateManagerInterface		
<b>Kind</b>	ProvidedPort	<b>Interface</b>	VehicleStateManagerInterface
<b>Description</b>	To receive the vehicle safety states from Vehicle State Manager Adaptive Application.		
<b>Variation</b>			

|(RS\_VUCM\_00037, RS\_VUCM\_00043)

Service Interface

**[SWS\_VUCM\_00183]{DRAFT} Definition of ServiceInterface VehicleStateManagerInterface** [

<b>Name</b>	VehicleStateManagerInterface
<b>Namespace</b>	ara::vucm

<b>Field</b>	CampaignState
<b>Description</b>	The current status of Campaign.
<b>Type</b>	CampaignStateType
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Field</b>	VehicleConditionCollection
<b>Description</b>	A set of safety conditions, for which values are available in the Vehicle Package, to be computed by the Vehicle State Manager Adaptive Application, along with its corresponding computed state.
<b>Type</b>	<a href="#">VehicleConditionCollectionType</a>
<b>HasGetter</b>	true
<b>HasNotifier</b>	true
<b>HasSetter</b>	false

<b>Method</b>	PublishSafetyState
<b>Description</b>	Method called by Vehicle State Manager Adaptive Application when safety state is changed
<b>FireAndForget</b>	false
<b>Parameter</b>	safetyStates
	<b>Description</b> Safety conditions computed by the Vehicle State Manager Adaptive Application.
	<b>Type</b> <a href="#">VehicleConditionCollectionType</a>
	<b>Variation</b>
	<b>Direction</b> IN

<b>Method</b>	VehicleCheck
<b>Description</b>	Method for Vehicle State Manager to inform V-UCM of vehicle check resolution
<b>FireAndForget</b>	false
<b>Parameter</b>	vehicleCheckResolution
	<b>Description</b> Vehicle check resolution. True if check succeeded.
	<b>Type</b> bool
	<b>Variation</b>
	<b>Direction</b> IN

]([RS\\_VUCM\\_00035](#), [RS\\_VUCM\\_00037](#))

### 9.3 Required Interface

[SWS\_VUCM\_01021]{DRAFT} Definition of Port PackageManagement required by functional cluster VUCM [

<b>Name</b>	PackageManagement
<b>Kind</b>	RequiredPort
	<b>Interface</b> PackageManagement
<b>Description</b>	Provides for the Adaptive Platform Machine services like receiving, processing and activating Software Packages. Also providing update history, status and Software Package and Software Cluster information.
<b>Variation</b>	

]()

## 9.4 Application Errors

### 9.4.1 Application Error Domain

#### 9.4.1.1 UCMErrDomain

This section lists all application errors of the [V-UCM](#).



**[SWS\_VUCM\_00136]{DRAFT} Definition of Application Error Domain of functional cluster VUCM** [

<i>Name</i>	<i>Code</i>	<i>Description</i>
kAuthenticationFailed	8	Package authentication failed.
kBlockInconsistent	25	Consistency check for transferred block failed.
kBusyWithCampaign	34	Campaign has already started.
kCancelFailed	16	Cancel failed.
kIncompatibleDelta	29	Delta package dependency check failed.
kIncompatiblePackageVersion	24	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or V-UCM.
kIncorrectBlock	2	The same block number is received twice.
kIncorrectBlockSize	30	The size of the block exceeds the provided block size from TransferStart or TransferVehiclePackage.
kIncorrectSize	3	The size of the Software or Vehicle Package exceeds the provided size in TransferStart.
kInsufficientData	6	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
kInsufficientMemory	1	Insufficient memory to perform operation.
kInvalidChecksumDescription	35	Checksum attribute not recognised.
kInvalidPackageManifest	13	Package manifest could not be read.
kInvalidTransferId	4	The Transfer ID is invalid.
kMissingDependencies	21	Activation is not allowed because dependencies are missing.
kNewCampaignDisabled	31	New campaigns are disabled, calling AllowCampaign will enable new campaigns.
kNotAbleToRevertPackages	15	RevertProcessedSwPackages failed.
kOldVersion	9	Software Package version is too old.
kOperationNotPermitted	5	The operation is not supported in the current context.
kPackageInconsistent	7	Package integrity check failed.
kPrepareUpdateFailed	19	Error during update preparation step.
kProcessSwPackageCancelled	22	The processing operation has been interrupted by a Cancel() call.
kProcessedSoftwarePackageInconsistent	23	The processed Software Package integrity check has failed.
kServiceBusy	12	Another processing is already ongoing and therefore the current processing request has to be rejected.
kSoftwareClusterMissing	37	The Software Cluster is not present in the Machine.
kSwclRemovalDenied	39	Attempt to remove PLATFORM_CORE Software Cluster.
kTransferFailed	38	UCM cannot persist transferred block.
kUnexpectedPackage	32	The Software Package name does not correspond to the RequestedPackage field value.
kUnsupportedPackageFormat	40	The Vehicle Package or Software Package archiving format is not supported.
kUpdateSessionRejected	33	Start of an update session was rejected by State Management
kVerificationFailed	36	State Management returned verification failure

] ([RS\\_VUCM\\_00039](#), [RS\\_VUCM\\_00043](#))

## 10 Configuration

The configuration structure of Vehicle Update And Configuration Management is described in [7] by Platform Module Development / Update and Configuration Management. This chapter defines default values and semantic constraints for this configuration model.

### 10.1 Default Values

This section defines the default values for attributes defined in [7].

### 10.2 Semantic Constraints

This section defines semantic constraints for the configuration elements of Vehicle Update and Configuration Management defined in [7].

## A Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Chapter is generated.

<b>Class</b>	<b>CryptoServiceCertificate</b>			
<b>Package</b>	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
<b>Note</b>	This meta-class represents the ability to model a cryptographic certificate. <b>Tags:</b> atp.recommendedPackage=CryptoServiceCertificates			
<b>Base</b>	<i>ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDesignElement, UploadablePackageElement</i>			
<b>Aggregated by</b>	ARPackage.element			
<b>Attribute</b>	<b>Type</b>	<b>Mult.</b>	<b>Kind</b>	<b>Note</b>
algorithmFamily	CryptoCertificate AlgorithmFamilyEnum	0..1	attr	This attribute represents a description of the family of crypto algorithm used to generate public key and signature of the cryptographic certificate.
format	CryptoCertificateFormat Enum	0..1	attr	This attribute can be used to provide information about the format used to create the certificate
maximumLength	PositiveInteger	0..1	attr	This attribute represents the ability to define the maximum length of the certificate in bytes.
nextHigherCertificate	<a href="#">CryptoServiceCertificate</a>	0..1	ref	The reference identifies the next higher certificate in the certificate chain.
serverNameIdentification	String	0..1	attr	Server Name Indication (SNI) is needed if the IP address hosts multiple servers (on the same port), each of them using a different certificate.  If the client sends the SNI to the Server in the client hello, the server looks the SNI up in its certificate list and uses the certificate identified by the SNI.

**Table A.1: CryptoServiceCertificate**

<b>Class</b>	<b>Referrable</b> (abstract)			
<b>Package</b>	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
<b>Note</b>	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
<b>Base</b>	<i>ARObject</i>			
<b>Subclasses</b>	<i>AtpDefinition, BswDistinguishedPartition, BswModuleCallPoint, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, CppImplementationDataTypeContextTarget, DiagnosticEnvModeElement, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, HwDescriptionEntity, ImplementationProps, ModeTransition, MultilanguageReferrable, NmNetworkHandle, PncMappingIdent, SingleLanguageReferrable, SoConIPdulIdentifier, SocketConnectionBundle, SomeipRequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent</i>			
<b>Attribute</b>	<b>Type</b>	<b>Mult.</b>	<b>Kind</b>	<b>Note</b>
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference.  <b>Stereotypes:</b> atpIdentityContributor <b>Tags:</b> xml.enforceMinMultiplicity=true xml.sequenceOffset=-100





<b>Class</b>	<b>Referrable</b> (abstract)			
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. <b>Tags:</b> xml.sequenceOffset=-90

**Table A.2: Referrable**

<b>Class</b>	<b>SoftwareCluster</b>			
<b>Package</b>	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
<b>Note</b>	This meta-class represents the ability to define an uploadable software-package, i.e. the SoftwareCluster shall contain all software and configuration for a given purpose. <b>Tags:</b> atp.recommendedPackage=SoftwareClusters			
<b>Base</b>	ARElement, ARObjct, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDeploymentElement, UploadablePackageElement			
<b>Aggregated by</b>	ARPackage.element			
<b>Attribute</b>	<b>Type</b>	<b>Mult.</b>	<b>Kind</b>	<b>Note</b>
artifact Checksum	ArtifactChecksum	*	aggr	This aggregation carries the checksums for artifacts contained in the enclosing SoftwareCluster. Please note that the value of these checksums is only applicable at the time of configuration. <b>Stereotypes:</b> atpSplitable <b>Tags:</b> atp.Splitkey=artifactChecksum.shortName, artifactChecksum.uri
artifactLocator	ArtifactLocator	*	aggr	This aggregation represents the artifact locations that are relevant in the context of the enclosing SoftwareCluster
claimed FunctionGroup	ModeDeclarationGroup Prototype	*	ref	Each SoftwareCluster can reserve the usage of a given functionGroup such that no other SoftwareCluster is allowed to use it
conflictsTo	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation handles conflicts. If it yields true then the SoftwareCluster shall not be installed. <b>Stereotypes:</b> atpSplitable <b>Tags:</b> atp.Splitkey=conflictsTo
contained ARElement	ARElement	*	ref	This reference represents the collection of model elements that cannot derive from UploadablePackageElement and that contribute to the completeness of the definition of the SoftwareCluster. <b>Stereotypes:</b> atpSplitable <b>Tags:</b> atp.Splitkey=containedARElement
containedFibex Element	FibexElement	*	ref	This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster.
contained Package Element	UploadablePackageElement	*	ref	This reference identifies model elements that are required to complete the manifest content. <b>Stereotypes:</b> atpSplitable <b>Tags:</b> atp.Splitkey=containedPackageElement
contained Process	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster.
dependsOn	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation can be taken to identify a dependency for the enclosing SoftwareCluster. <b>Stereotypes:</b> atpSplitable <b>Tags:</b> atp.Splitkey=dependsOn





Class	SoftwareCluster			
design	SoftwareClusterDesign	*	ref	This reference represents the identification of all SoftwareClusterDesigns applicable for the enclosing SoftwareCluster. <b>Stereotypes:</b> atpUriDef
diagnosticDeploymentProps	SoftwareClusterDiagnosticDeploymentProps	0..1	ref	This reference identifies the applicable SoftwareClusterDiagnosticDeploymentProps that are applicable for the referencing SoftwareCluster.
installationBehavior	SoftwareClusterInstallationBehaviorEnum	0..1	attr	This attribute controls the behavior of the SoftwareCluster in terms of installation.
license	Documentation	*	ref	This attribute allows for the inclusion of the full text of a license of the enclosing SoftwareCluster. In many cases open source licenses require the inclusion of the full license text to any software that is released under the respective license.
moduleInstantiation	AdaptiveModuleInstantiation	*	ref	This reference identifies AdaptiveModuleInstantiations that need to be included with the SoftwareCluster in order to establish infrastructure required for the installation of the SoftwareCluster. <b>Stereotypes:</b> atpSplitable <b>Tags:</b> atp.Splitkey=moduleInstantiation
releaseNotes	Documentation	0..1	ref	This attribute allows for the explanations of changes since the previous version. The list of changes might require the creation of multiple paragraphs of text.
typeApproval	String	0..1	attr	This attribute carries the homologation information that may be specific for a given country.
vendorId	PositiveInteger	0..1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list.
vendorSignature	<a href="#">CryptoServiceCertificate</a>	0..1	ref	This reference identifies the certificate that represents the vendor's signature.
version	<a href="#">StrongRevisionLabelString</a>	0..1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster.

**Table A.3: SoftwareCluster**

Class	SoftwarePackage			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to formalize the content of a software package. <b>Tags:</b> atp.recommendedPackage=SoftwarePackages			
Base	<i>ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDeploymentElement, UploadablePackageElement</i>			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
actionType	SoftwarePackageActionTypeEnum	0..1	attr	This attribute defines the action to be taken in the step of processing the enclosing SoftwarePackage.
activationAction	SoftwarePackageActivationActionEnum	0..1	attr	This attribute governs the action to be taken after the installation of the SoftwareCluster completed.
compressedSoftwarePackageSize	PositiveInteger	0..1	attr	This size represents the size of the compressed SoftwarePackage.





Class	SoftwarePackage			
deltaPackage Applicable Version	<a href="#">StrongRevisionLabelString</a>	0..1	attr	This attribute identifies the version of the included SoftwareCluster for which the enclosing SoftwarePackage can be used as a delta update
estimated DurationOf Operation	TimeValue	0..1	attr	This attribute provides an estimation about how long the operation of the SoftwarePackage is going to take for its transfer, processing and activation when updated standalone (not within an update campaign)
minimum SupportedUcm Version	RevisionLabelString	0..1	attr	This attribute identifies the minimum supported version of the UCM for this SoftwarePackage.
packagerId	PositiveInteger	0..1	attr	This attribute identifies Id of the organization that provides the packager generating the SoftwarePackage.
packager Signature	<a href="#">CryptoServiceCertificate</a>	0..1	ref	This reference identifies the certificate that represents the packager's signature.
purposeOf Update	Documentation	0..1	ref	The referenced Documentation is supposed to provide a description of the purpose of the update.
softwareCluster	<a href="#">SoftwareCluster</a>	0..1	ref	This reference identifies the SoftwareCluster that belongs to the SoftwarePackage. The nature of this relation is actually more like an aggregation than a reference. But the relation is still modelled as a reference because two ARElements cannot aggregate each other.
uncompressed SoftwareCluster Size	PositiveInteger	0..1	attr	This attribute gives an indication about the storage that has to be available on the target.

**Table A.4: SoftwarePackage**

Primitive	StrongRevisionLabelString
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	<p>This primitive represents a revision label which identifies an object under version control. It represents a pattern which requires three integer numbers separated by a dot, representing from left to right Major Version, MinorVersion, PatchVersion and additional labels for pre-release version and build metadata.</p> <p>Legal patterns are for example: 1.0.0-alpha+001 1.0.0+20130313144700 1.0.0-beta+exp.sha.5114f85</p> <p><b>Tags:</b>  xml.xsd.customType=STRONG-REVISION-LABEL-STRING  xml.xsd.pattern=(0 [1-9]d*)\.(0 [1-9]d*)\.(0 [1-9]d*)(-((0 [1-9]d*)\d*[a-zA-Z][0-9a-zA-Z-]*)\.(0 [1-9]d*)\d*[a-zA-Z][0-9a-zA-Z-]*)*)?(\+([0-9a-zA-Z-]+\.[0-9a-zA-Z-]+)*)?  xml.xsd.type=string</p>

**Table A.5: StrongRevisionLabelString**

Class	UcmModuleInstantiation (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Ucm			
Note	This meta-class represents the ability to define the deployment of a UCM instantiation.			
Base	<i>ARObject</i> , <i>AdaptiveModuleInstantiation</i> , <i>AtpClassifier</i> , <i>AtpFeature</i> , <i>AtpStructureElement</i> , <i>Identifiable</i> , <i>MultilanguageReferrable</i> , <i>NonOsModuleInstantiation</i> , <a href="#">Referrable</a>			
Subclasses	UcmMasterModuleInstantiation, UcmSubordinateModuleInstantiation			
Aggregated by	<i>AtpClassifier</i> . <i>atpFeature</i> , <i>Machine</i> . <i>moduleInstantiation</i>			
Attribute	Type	Mult.	Kind	Note
identifier	String	0..1	attr	This represents the identification of a UCM.
maxBlockSize	PositiveInteger	0..1	attr	This attribute denotes the maximum block size (unit: bytes) used in the UCM implementation.





<b>Class</b>	<b>UcmModuleInstantiation</b> (abstract)			
version	<a href="#">StrongRevisionLabelString</a>	0..1	attr	This attribute defines the software version of the UCM on this platform.  Note that the definition of the version is required if the ability of the SoftwarePackage to require a minimum version of the UCM is utilized.

**Table A.6: UcmModuleInstantiation**

<b>Class</b>	<b>UcmRetryStrategy</b>			
<b>Package</b>	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Ucm			
<b>Note</b>	This meta-class describes the configuration of the retry strategy for a sub-class of UcmModule Implementation.			
<b>Base</b>	<a href="#">ARObject</a> , <a href="#">Identifiable</a> , <a href="#">MultilanguageReferrable</a> , <a href="#">Referrable</a>			
<b>Aggregated by</b>	UcmMasterModuleInstantiation.blockInconsistent, UcmMasterModuleInstantiation.serviceBusy, UcmMasterModuleInstantiation.updateSessionRejected, UcmSubordinateModuleInstantiation.prepareRollback, UcmSubordinateModuleInstantiation.prepareUpdate, UcmSubordinateModuleInstantiation.verifyUpdate			
<b>Attribute</b>	<b>Type</b>	<b>Mult.</b>	<b>Kind</b>	<b>Note</b>
maximumNumberOfRetries	PositiveInteger	0..1	attr	This attribute defines the maximum number of time the UCM module instantiation shall attempt a retry.
retryIntervalTime	TimeValue	0..1	attr	This attribute defines the time (in seconds) between two retry attempts.

**Table A.7: UcmRetryStrategy**

<b>Class</b>	<b>UcmToTimeBaseResourceMapping</b>			
<b>Package</b>	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Ucm			
<b>Note</b>	This meta-class maps the UCM Module Instantiation to the TimeSync Module Instantiation. <b>Tags:</b> atp.recommendedPackage=FCInteractions			
<b>Base</b>	<a href="#">ARElement</a> , <a href="#">ARObject</a> , <a href="#">CollectableElement</a> , <a href="#">FunctionalClusterInteractsWithFunctionalClusterMapping</a> , <a href="#">Identifiable</a> , <a href="#">MultilanguageReferrable</a> , <a href="#">PackageableElement</a> , <a href="#">Referrable</a> , <a href="#">UploadableDeploymentElement</a> , <a href="#">UploadablePackageElement</a>			
<b>Aggregated by</b>	ARPackage.element			
<b>Attribute</b>	<b>Type</b>	<b>Mult.</b>	<b>Kind</b>	<b>Note</b>
timeBaseResource	TimeBaseResource	0..1	ref	This reference identifies the relevant TimeBaseResource.
ucm	<a href="#">UcmModuleInstantiation</a>	0..1	ref	This reference identifies the relevant UcmModule Instantiation.

**Table A.8: UcmToTimeBaseResourceMapping**

<b>Class</b>	<b>VehicleDriverNotification</b>			
<b>Package</b>	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
<b>Note</b>	This meta-class provides the ability to configure a notification of the vehicle driver with respect to the update of vehicle software.			
<b>Base</b>	<a href="#">ARObject</a>			
<b>Aggregated by</b>	<a href="#">VehiclePackage.driverNotification</a>			
<b>Attribute</b>	<b>Type</b>	<b>Mult.</b>	<b>Kind</b>	<b>Note</b>





Class	VehicleDriverNotification			
approvalRequired	Boolean	0..1	attr	This attribute controls whether approval is required for the driver notification.
notificationState	VehicleDriverNotificationEnum	0..1	attr	This attribute is used to configure the notification state.

**Table A.9: VehicleDriverNotification**

Class	VehiclePackage			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to define a vehicle package for executing an update campaign. <b>Tags:</b> atp.recommendedPackage=VehiclePackages			
Base	ARElement, ARObject, CollectableElement, Identifiable, MultilanguageReferrable, PackageableElement, Referrable, UploadableDeploymentElement, UploadablePackageElement			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
driverNotification	<a href="#">VehicleDriverNotification</a>	*	aggr	This aggregation provides the ability to configure the necessary driver notifications.
estimatedDurationOfCampaign	TimeValue	0..1	attr	This attribute provides an estimation about how long the campaign based on the VehiclePackage is going to take.
maximumDurationOfCampaign	TimeValue	0..1	attr	Maximum time allowed for the campaign to be active until UCM Master automatically cancels the campaign.
minimumSupportedUcmMasterVersion	RevisionLabelString	0..1	attr	This attribute identifies the minimum supported version of the UCM Master for this VehiclePackage.
packagerSignature	<a href="#">CryptoServiceCertificate</a>	0..1	ref	This reference identifies the certificate that represents the packager's signature.
repository	UriString	0..1	attr	This attribute identifies the repository where the Vehicle Package is stored.
rolloutQualification(ordered)	<a href="#">VehicleRolloutStep</a>	*	aggr	This represents the rollout qualification.
ucm	UcmDescription	*	aggr	This aggregation represents the UcmDescriptions to be considered in the context of the VehiclePackage.
ucmMasterFallback(ordered)	UcmDescription	*	ref	This reference lists the fallback order of Ucms that can take over the master role if the master goes down.
vehicleDescription	Documentation	0..1	ref	This reference identifies the vehicle description.

**Table A.10: VehiclePackage**

Class	VehicleRolloutStep			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to define a rollout-condition for a vehicle update campaign.			
Base	ARObject, Identifiable, MultilanguageReferrable, Referrable			
Aggregated by	<a href="#">VehiclePackage.rolloutQualification</a>			
Attribute	Type	Mult.	Kind	Note







<i>Class</i>	<b>VehicleRolloutStep</b>			
safetyCondition	String	*	attr	This attribute represents a list of textual safety conditions (e.g.: close the driver window) that need to be fulfilled before the rollout step can proceed and need to be maintained while the campaign's rolloutQualification is executed.
ucmProcessing	UcmStep	*	aggr	This aggregation collects the UcmProcessingSteps that make up the rollout step.
violatedSafetyConditionBehavior	<a href="#">ViolatedSafetyConditionBehaviorEnum</a>	0..1	attr	This attribute provides options for the configuration of the reaction to a violated safety condition.

**Table A.11: VehicleRolloutStep**

<i>Enumeration</i>	<b>ViolatedSafetyConditionBehaviorEnum</b>
<i>Package</i>	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution
<i>Note</i>	This enumeration provides formal options for the configuration of the reaction to a violated safety condition.
<i>Aggregated by</i>	<a href="#">VehicleRolloutStep.violatedSafetyConditionBehavior</a>
<i>Literal</i>	<b>Description</b>
cancelCampaign	This enumerator supports the ability to cancel the update campaign in response to a violated safety condition. <b>Tags:</b> atp.EnumerationLiteralIndex=1
waitForVehicleSafeState	This enumerator supports the ability to wait for the vehicle to acquire a safe state in response to a violated safety condition. <b>Tags:</b> atp.EnumerationLiteralIndex=0

**Table A.12: ViolatedSafetyConditionBehaviorEnum**

## **B Platform Extension API (normative)**

The focus of the APIs in this section are for OEM-specific platform extensions. The abstraction of the interfaces is lower which could lead to a higher machine dependency.

## **C Interfaces to other Functional Clusters (informative)**

### **C.1 Overview**

AUTOSAR decided not to standardize interfaces which are exclusively used between Functional Clusters (on platform-level only), to allow efficient implementations, which might depend e.g. on the used Operating System.

This chapter provides informative guidelines how the interaction between Functional Clusters looks like, by clustering the relevant requirements of this document to describe Inter-Functional Cluster (IFC) interfaces. In addition, the standardized public interfaces which are accessible by user space applications (see chapters [8](#) and [9](#)) can also be used for interaction between Functional Clusters.

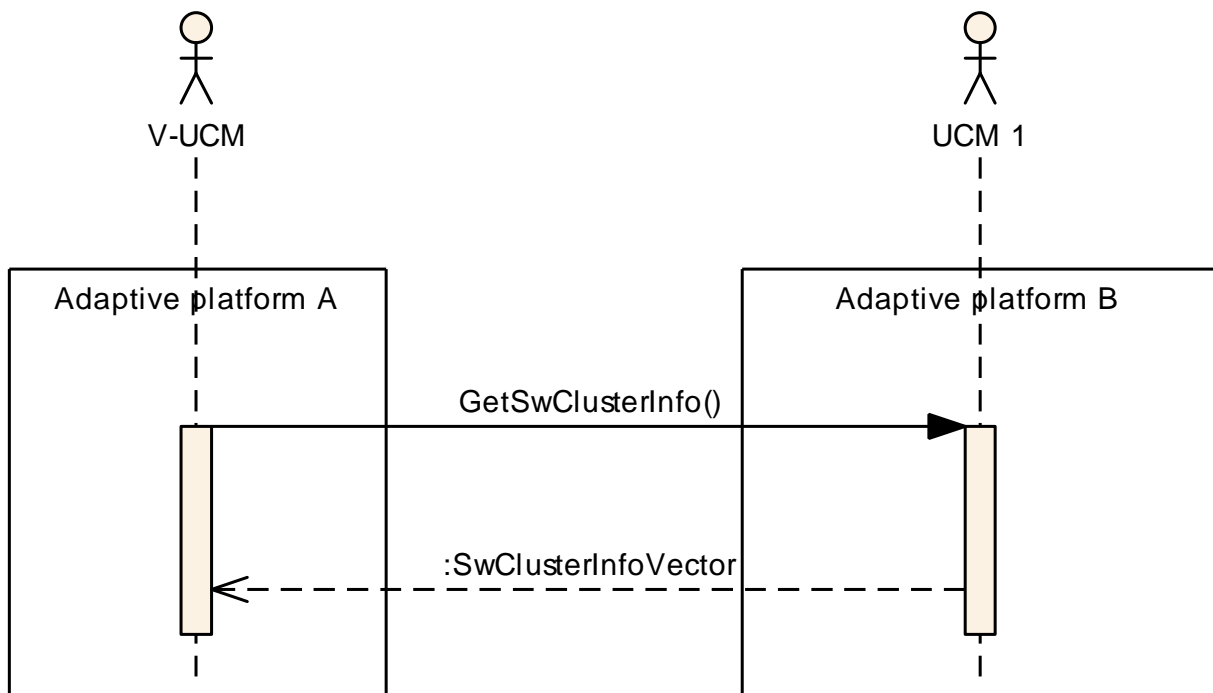
The goal is to provide a clear understanding of Functional Cluster boundaries and interaction, without specifying syntactical details. This ensures compatibility between documents specifying different Functional Clusters and supports parallel implementation of different Functional Clusters. Details of the interfaces are up to the platform provider. Additional interfaces, parameters and return values can be added.

### **C.2 Interface Tables**

## D Packages distribution within vehicle detailed sequence examples

### D.1 Collect information of present Software Clusters in vehicle

From a regular basis, **V-UCM** and **UCM** can collect information of present **Software Clusters** from the other **AUTOSAR Adaptive Platforms** of the vehicle in order to be used later when communicating with **Backend** and then determine if there are new actions (update, remove, install) required.



**Figure D.1: Collect information of Software Clusters present in vehicle from several AUTOSAR Adaptive Platforms**

### D.2 Action computation

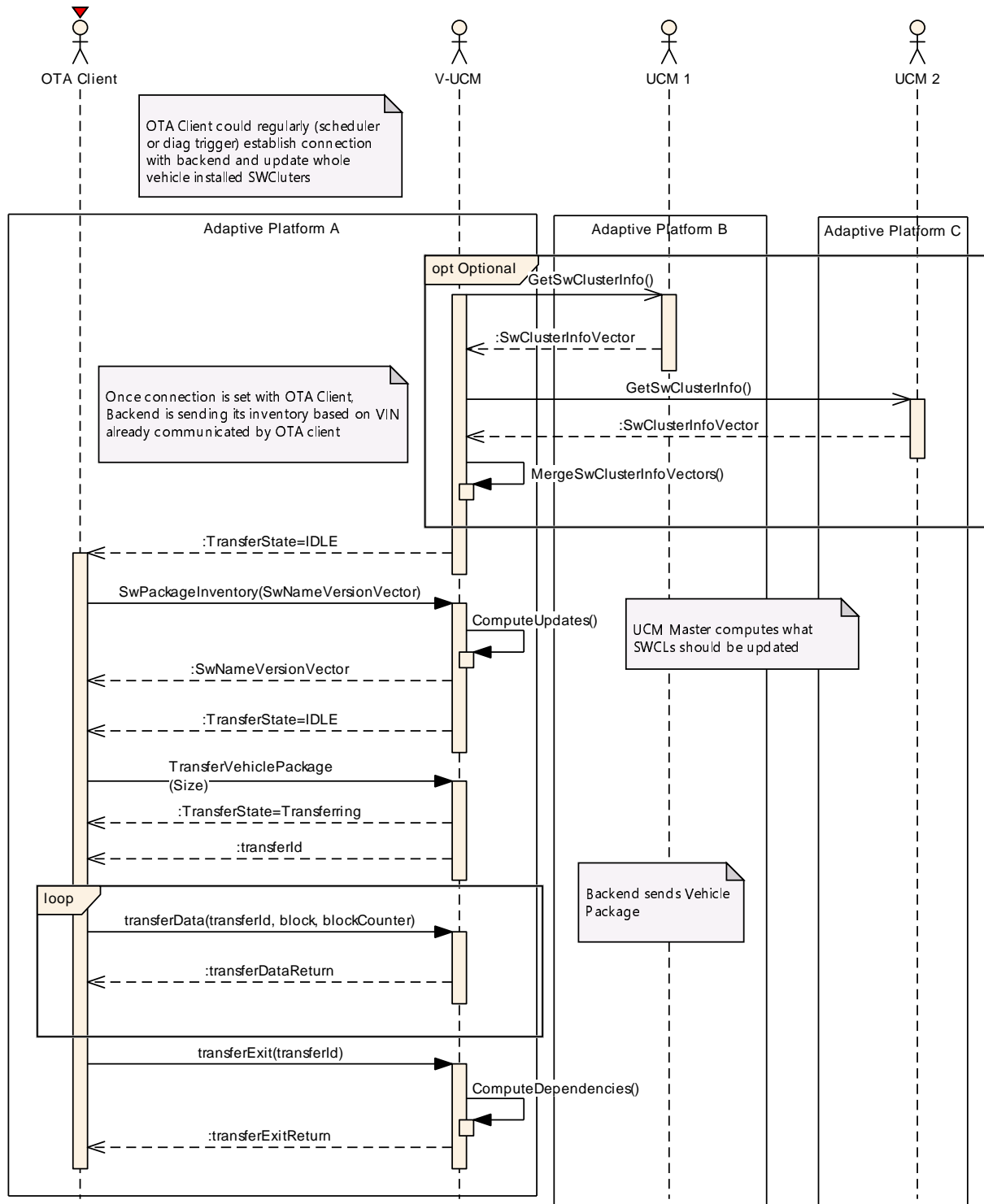
In order to find out if there is a new update available from **Backend** or the need to install or remove a **Software Cluster**, vehicle and **Backend** have to share their current status and either **Backend** or vehicle have to compute what **UCM Master** actions are needed.

**Backend** will have the possibility to push a package into the vehicle when communication is established, for instance for security purpose.

Communication trial between **Backend** and **V-UCM** can be done on driver’s request or from a scheduler.

### D.2.1 Pull package from Backend into vehicle

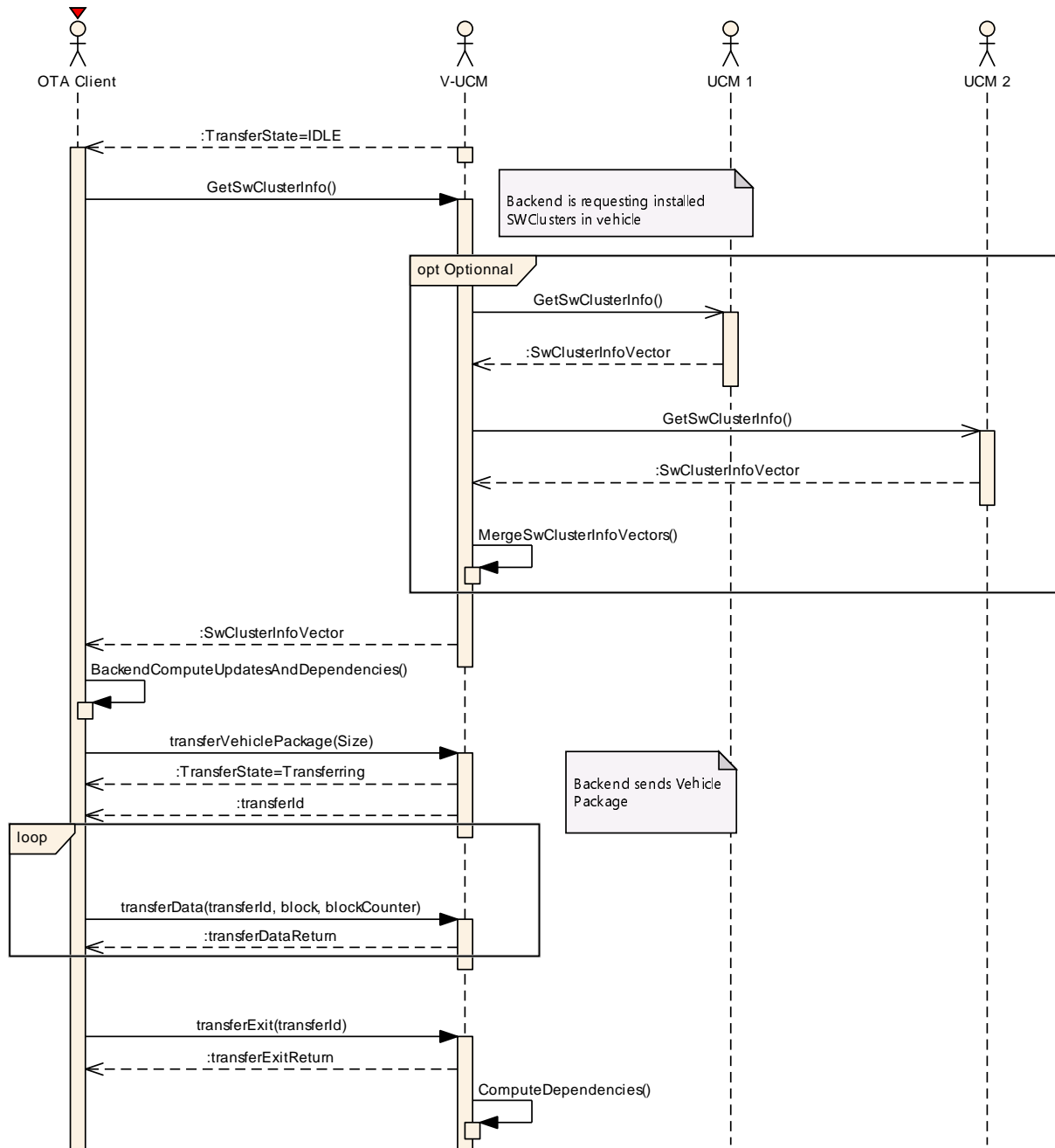
Case where vehicle is computing the difference between **Software Clusters** versions that are present in vehicle and the ones available in **Backend**.



**Figure D.2: Pull package from backend**

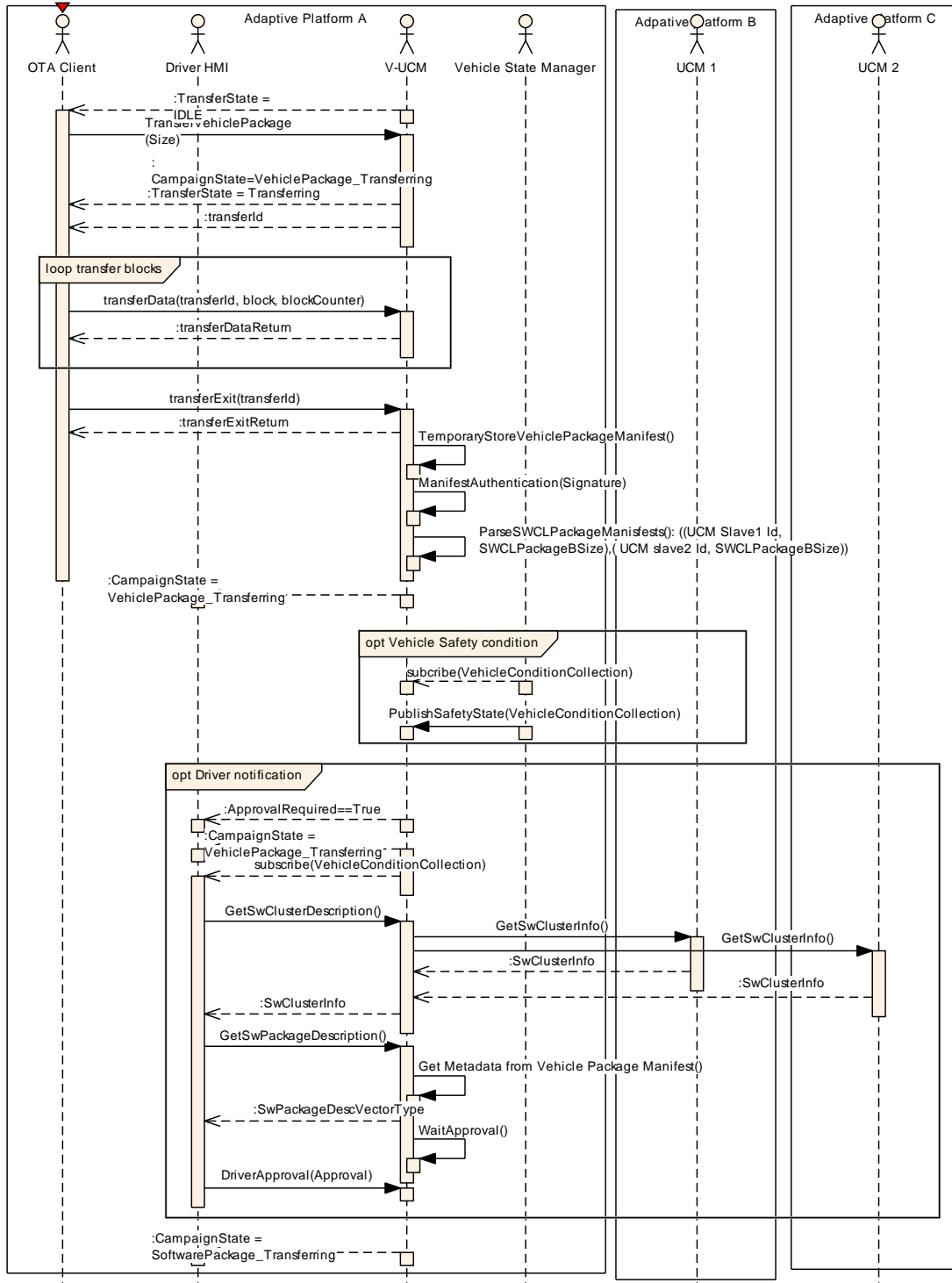
### D.2.2 Push package from backend into vehicle

Case where Backend is computing the difference between Software Clusters versions that are present in vehicle and the ones available in Backend.



**Figure D.3: Push package from backend**

### D.3 Packages transfer from backend into targeted UCM



**Figure D.4: Stream packages blocks from backend into targeted UCM**

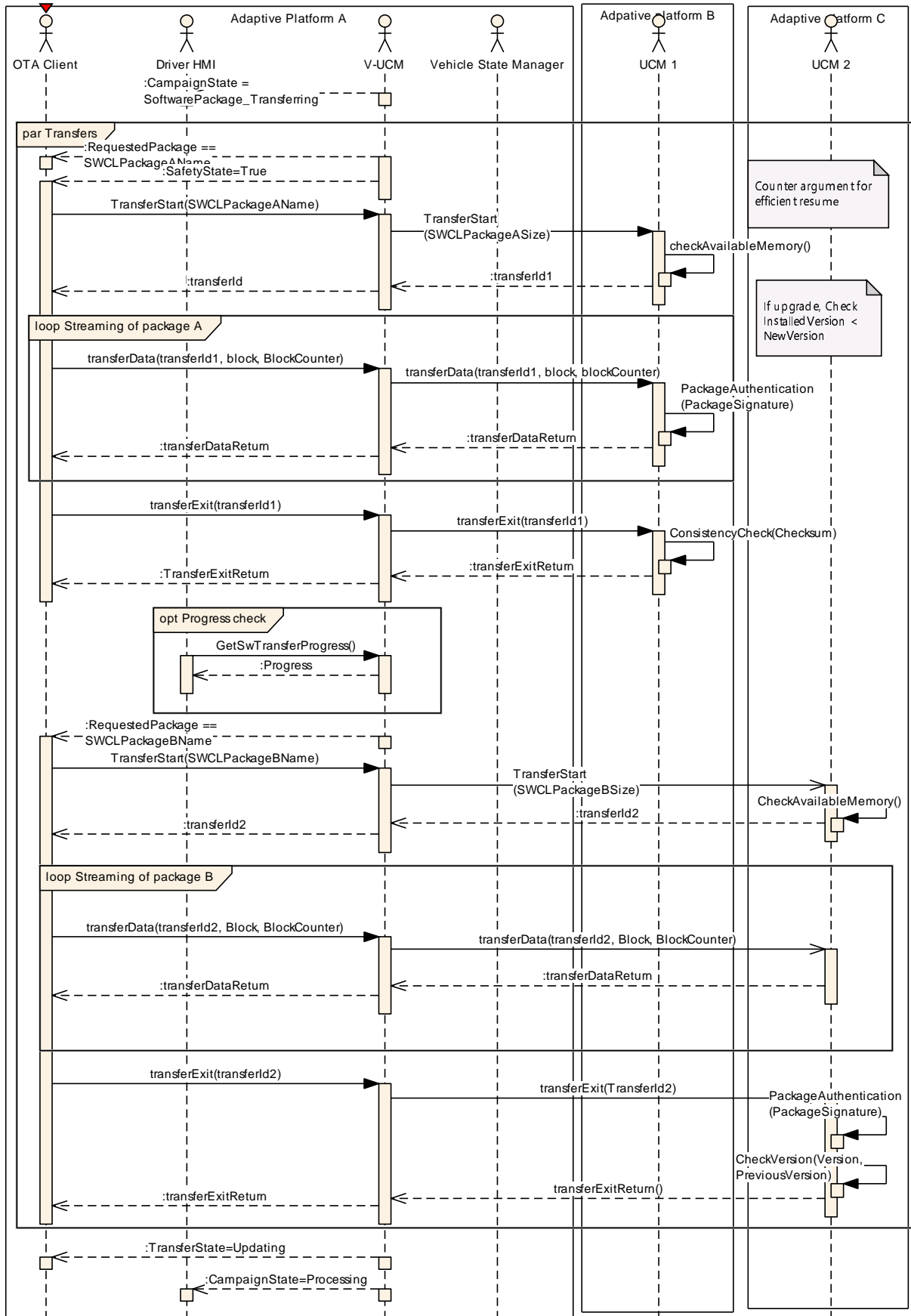
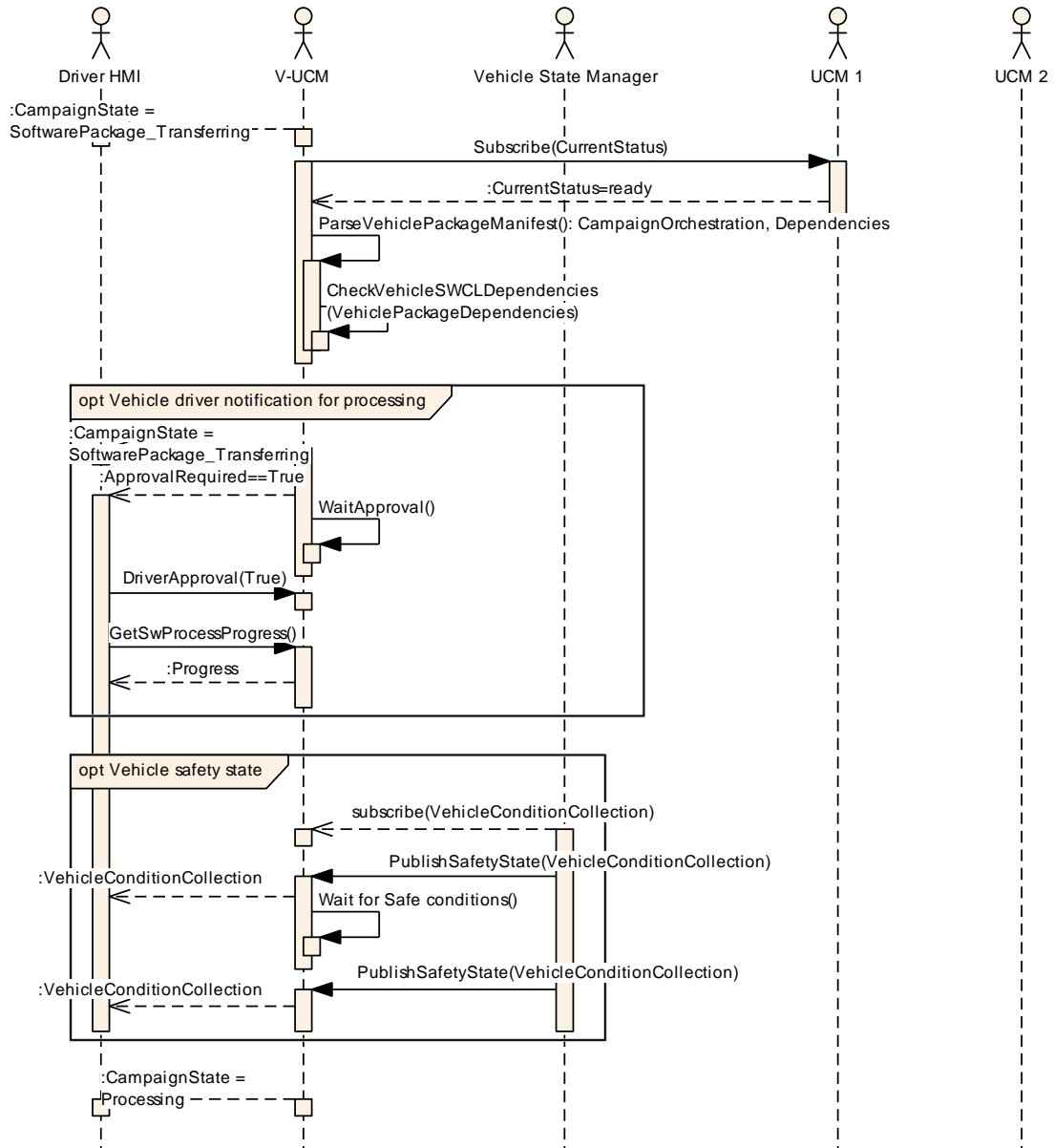


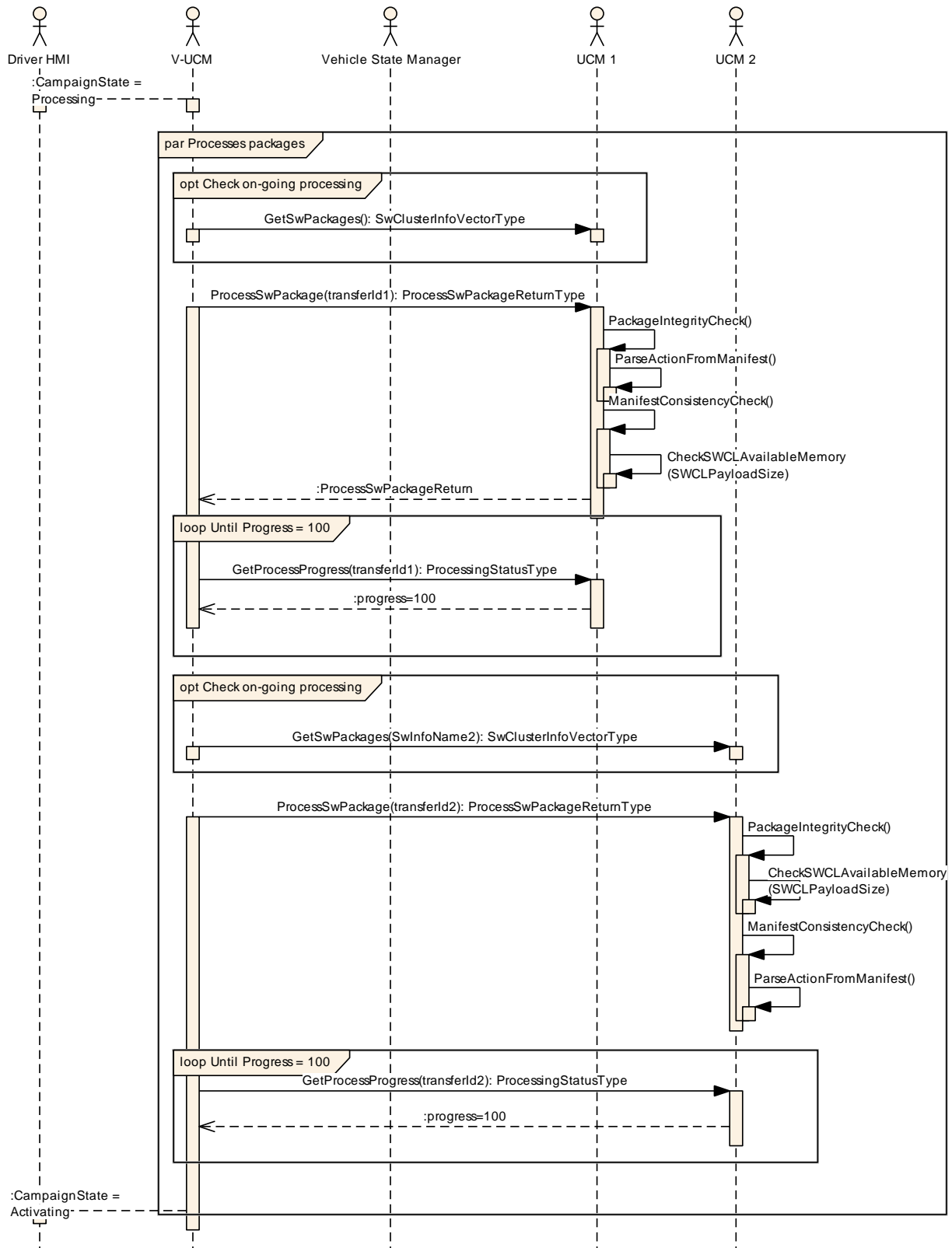
Figure D.5: Stream packages blocks from backend into targeted UCM



### D.4 Package processing



**Figure D.6: Packages processing by UCMs**



**Figure D.7: Packages processing by UCMs**

### D.5 Package activation

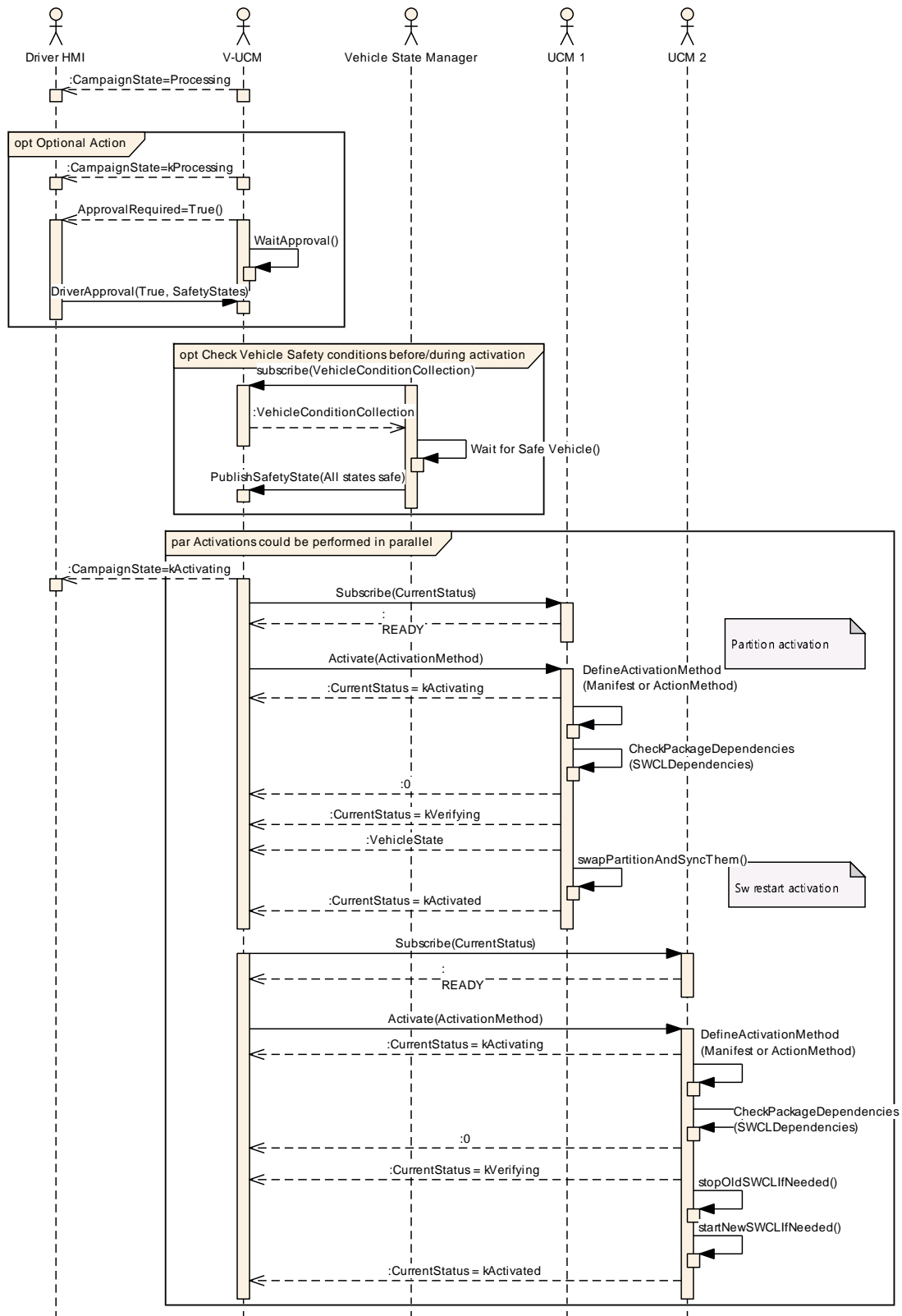
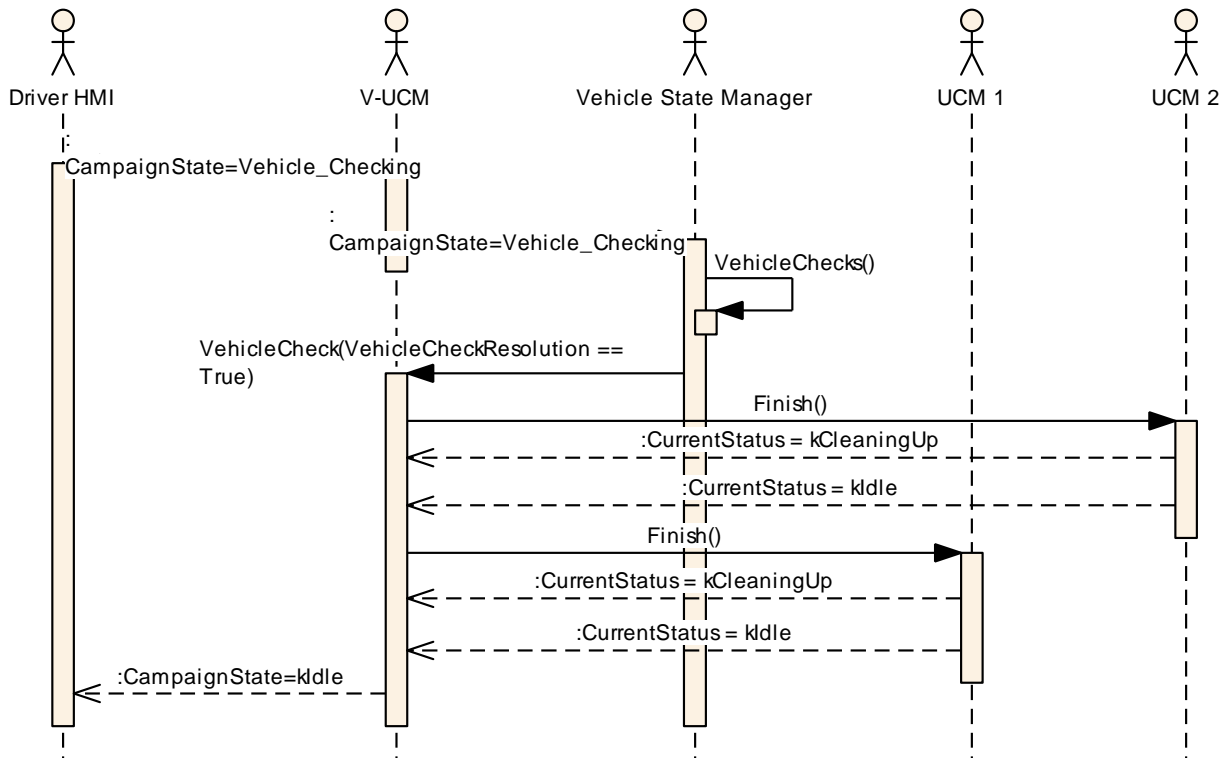
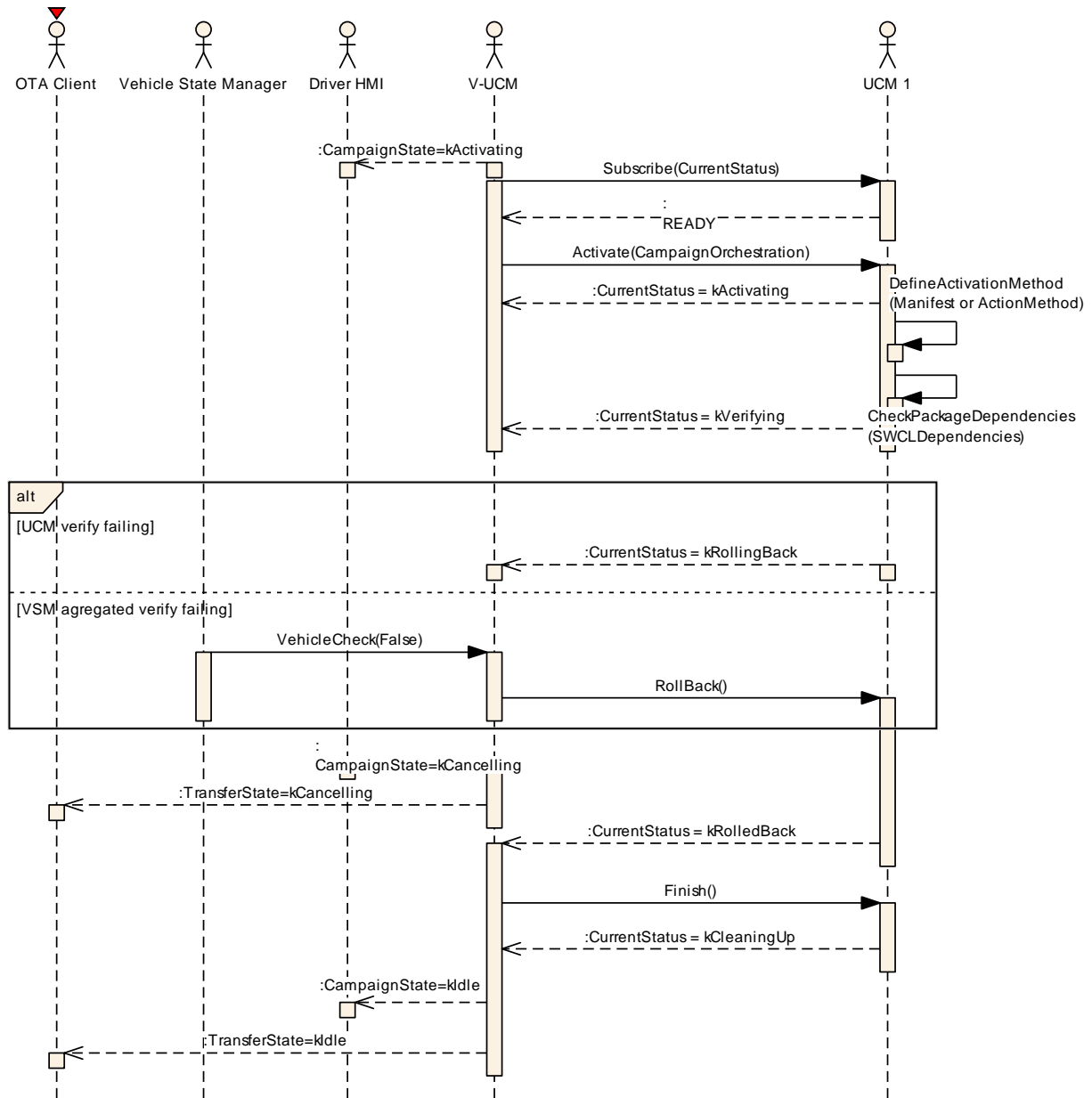


Figure D.8: Packages activation by UCMs



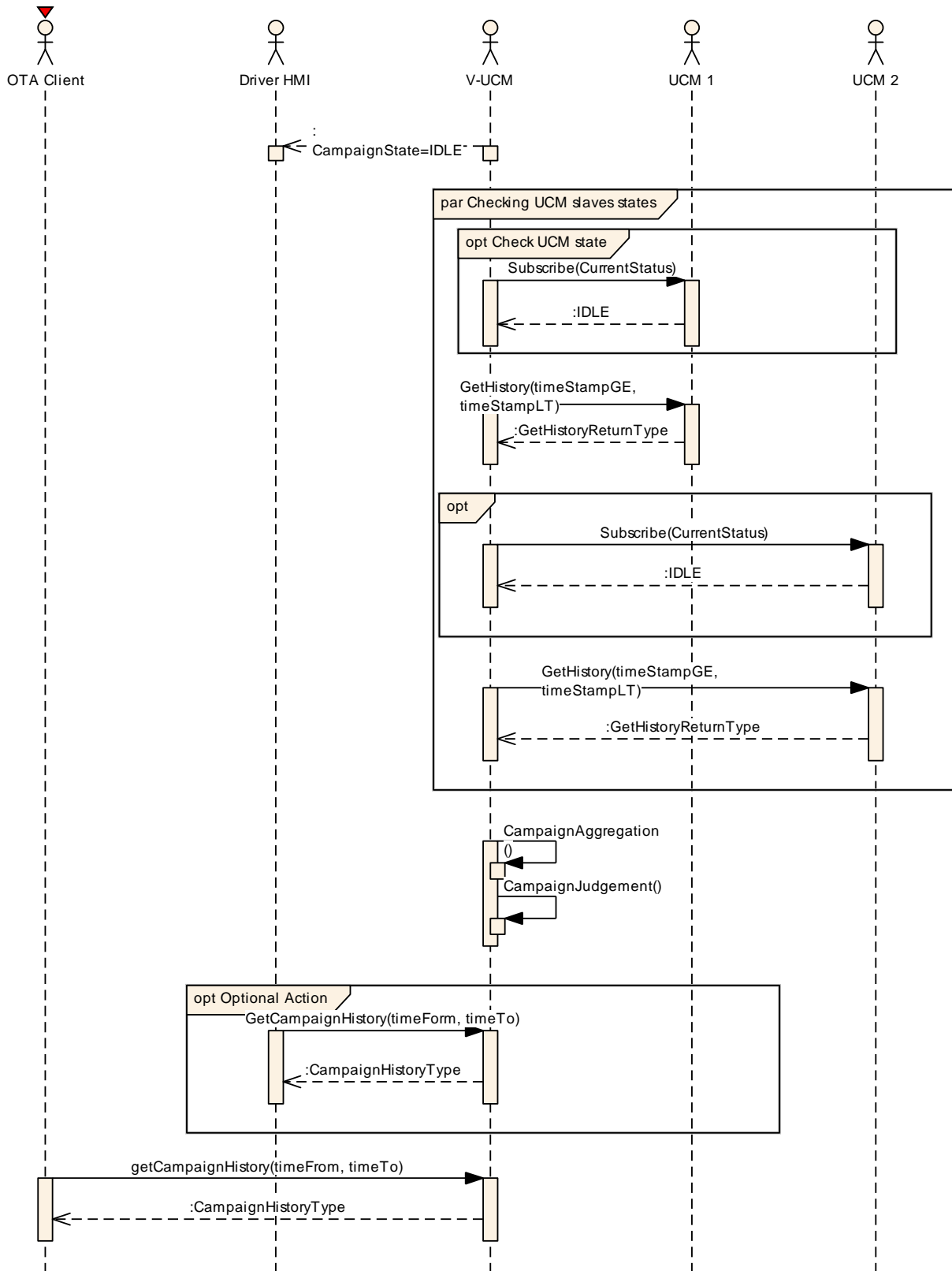
**Figure D.9: Packages activation by UCMs**

**D.6 Package rollback**



**Figure D.10: Packages rollback by UCMs**

### D.7 Campaign reporting



**Figure D.11: Campaign reporting to backend**

## E Security Analysis of Installation and Update

This chapter presents a summary for the security analysis of the V-UCM. Some of the threats could not be addressed by specifying AUTOSAR requirements. The main reason for not specifying the countermeasures is to allow vendors to flexibly decide on the solution that fits their setup. Here we aim to raise awareness and provide advice on the selected topics:

### E.1 Securing Vehicle Package

V-UCM is following update campaign contained in the Vehicle Packages it receives. Therefore, integrity and authenticity of Vehicle Package are critical to protect system integrity. It shall be ensured that the Vehicle Package is neither illegitimately altered nor issued by unauthorized parties. This can be achieved by applying cryptographic techniques such as digital signatures. The period that a Vehicle Package resides in V-UCM before the campaign being completed shall not be neglected. It provides a window of opportunity for an attacker to tamper with the Vehicle Package after the authentication is done at TransferExit.

Information disclosure is another security threat category that might be applicable to a Vehicle Package. If it contains sensitive information, such as intellectual properties or cryptographic keys, require confidentiality protection in addition to integrity and authenticity when being persisted or transmitted over a communication channel.

Another aspect of protecting a Vehicle Package is its freshness. An attacker may try to manipulate the system by downgrading the software via replaying an authentic but older Software Package. In this regard, the platform shall ensure that only newer packages (i.e. packages that contain newer version of installed SWCL) can be installed.

### E.2 Securing Calls to V-UCM

V-UCM provides a very critical functionality in the vehicle that allows modifying applications and platform components. In that sense, it is critical to prevent unauthorized access to V-UCM, meaning only legitimate callers should be allowed to reach the V-UCM service interfaces. This is primarily enforced in the communication layer supported by the Identity and Access Management. Additionally, the calls to the V-UCM interfaces shall be protected against altering, e.g. changing API arguments. When the service and client reside on the same machine, the security relies on the integrity of the operating system and the platform. In case, the service and the client are running on different machines, a secure communication, assuring authenticity and integrity of communication, is additionally required.

Moreover, some API methods of the V-UCM interfaces returns sensitive information about the platform. This subset (for instance GetSwClusterInfo, GetCampaignHistory,

GetSwPackages, SwPackageInventory) shall be protected against information disclosure and should only be reachable over a channel that provides confidentiality.

### **E.3 Suppressing Call to V-UCM**

Multiple scenarios can be envisioned where an attacker targets suppressing the calls to V-UCM. The attack could block the calls to or the response from V-UCM. In both cases the caller of the service may assume that V-UCM is not responding and retries its request. This would lead to undesired overhead on the system. For such scenarios, it is recommended that both V-UCM and the Adaptive Applications interacting with V-UCM consider reporting security events when same calls repeatedly received at V-UCM or calls repeatedly fail at the caller side. This information could potentially be picked up by Intrusion Detection Systems or Anomaly Detection Systems.

### **E.4 Resource Starvation**

According to the current specification, the available resources for transferring a Vehicle Package is only checked when TransferStart is called but not reserved. This means, while the transfer is ongoing, the system storage can be exhausted by other processes using the same storage media. In this regard, a solution could be to reserve the necessary resources for the Vehicle Package transfer or processing from the beginning to prevent attacks aiming at such scenarios.



## F Demonstrator Examples

Following is pdu api example header file:

```
1 #ifndef _PDUAPI_EXPORT
2 #   define EXTERNC extern "C"
3 #else // _PDUAPI_EXPORT
4 #   define EXTERNC /* EXTERNC */
5 #endif // _PDUAPI_EXPORT
6
7 typedef uint8_t UNUM8; /* Unsigned numeric 8 bits.*/
8 typedef int8_t SNUM8; /* Signed numeric 8 bits.*/
9 typedef uint16_t UNUM16; /* Unsigned numeric 16 bits.*/
10 typedef int16_t SNUM16; /* Signed numeric 16 bits.*/
11 typedef uint32_t UNUM32; /* Unsigned numeric 32 bits.*/
12 typedef int32_t SNUM32; /* Signed numeric 32 bits.*/
13 typedef char CHAR8; /* ASCII-coded 8-bit character value (ISO 8859-1 (Latin
14     1)).*/
15
16 constexpr UNUM32 PDU_ID_UNDEF = 0xFFFFFFFF; /* Undefined ID value. Used to
17     indicate
18     an ID value is undefined.*/
19
20 constexpr UNUM32 PDU_HANDLE_UNDEF = 0xFFFFFFFF; /* Undefined handle value.
21     Used to indicate a Handle value is
22     undefined. */
23
24 typedef enum E_PDU_IT
25 {
26     PDU_IT_IO_UNUM32 = 0x1000, /* IOCTL UNUM32 item. */
27     PDU_IT_IO_PROG_VOLTAGE = 0x1001, /* IOCTL Program Voltage item. */
28     PDU_IT_IO_BYTEARRAY = 0x1002, /* IOCTL Byte Array item. */
29     PDU_IT_IO_FILTER = 0x1003, /* IOCTL Filter item. */
30     PDU_IT_IO_EVENT_QUEUE_PROPERTY = 0x1004, /* IOCTL Event Queue Property
31     item. */
32     PDU_IT_RSC_STATUS = 0x1100, /* Resource Status item */
33     PDU_IT_PARAM = 0x1200, /* ComParam item */
34     PDU_IT_RESULT = 0x1300, /* Result item */
35     PDU_IT_STATUS = 0x1301, /* Status notification item */
36     PDU_IT_ERROR = 0x1302, /* Error notification item */
37     PDU_IT_INFO = 0x1303, /* Information notification item */
38     PDU_IT_RSC_ID = 0x1400, /* Resource ID item */
39     PDU_IT_RSC_CONFLICT = 0x1500, /* Resource Conflict Item */
40     PDU_IT_MODULE_ID = 0x1600, /* Module item */
41     PDU_IT_UNIQUE_RESP_ID_TABLE = 0x1700, /* Unique Response Id Table Item
42     */
43 } T_PDU_IT;
44
45 typedef enum E_PDU_COPT
46 {
47     PDU_COPT_STARTCOMM = 0x8001,
48     PDU_COPT_STOPCOMM = 0x8002,
49     PDU_COPT_UPDATEPARAM = 0x8003,
50     PDU_COPT_SENDRECV = 0x8004,
51     PDU_COPT_DELAY = 0x8005,
52     PDU_COPT_RESTORE_PARAM = 0x8006
53 } T_PDU_COPT;
```

```
48
49 typedef enum E_PDU_OBJT
50 {
51     PDU_OBJT_PROTOCOL = 0x8021, /* Object type for object PROTOCOL of MDF.
52     */
53     PDU_OBJT_BUSTYPE = 0x8022, /* Object type for object BUSTYPE of MDF.*/
54     PDU_OBJT_IO_CTRL = 0x8023, /* Object type for object IO_CTRL of MDF.*/
55     PDU_OBJT_COMPARAM = 0x8024, /* Object type for object COMPARAM of MDF.
56     */
57     PDU_OBJT_PINTYPE = 0x8025, /* Object type for object PINTYPE of MDF.*/
58     PDU_OBJT_RESOURCE = 0x8026 /* Object type for object RESOURCE of MDF.*/
59 } T_PDU_OBJT;
60
61 typedef enum E_PDU_STATUS
62 {
63     /* ComPrimitive status */
64     PDU_COPST_IDLE = 0x8010,
65     PDU_COPST_EXECUTING = 0x8011,
66     PDU_COPST_FINISHED = 0x8012,
67     PDU_COPST_CANCELLED = 0x8013,
68     PDU_COPST_WAITING = 0x8014,
69     /* ComLogicalLink status */
70     PDU_CLLST_OFFLINE = 0x8050,
71     PDU_CLLST_ONLINE = 0x8051,
72     PDU_CLLST_COMM_STARTED = 0x8052,
73     /* Module status */
74     PDU_MODST_READY = 0x8060,
75     PDU_MODST_NOT_READY = 0x8061,
76     PDU_MODST_NOT_AVAIL = 0x8062,
77     PDU_MODST_AVAIL = 0x8063,
78 } T_PDU_STATUS;
79
80 typedef enum E_PDU_INFO
81 {
82     PDU_INFO_MODULE_LIST_CHG = 0x8070,
83     PDU_INFO_RSC_LOCK_CHG = 0x8071,
84     PDU_INFO_PHYS_COMPARAM_CHG = 0x8072
85 } T_PDU_INFO;
86
87 typedef enum E_PDU_EVT_DATA
88 {
89     PDU_EVT_DATA_AVAILABLE = 0x0801,
90     PDU_EVT_DATA_LOST = 0x0802
91 } T_PDU_EVT_DATA;
92
93 typedef enum E_PDU_FILTER
94 {
95     PDU_FLT_PASS = 0x00000001,
96     PDU_FLT_BLOCK = 0x00000002,
97     PDU_FLT_PASS_UUDT = 0x00000011,
98     PDU_FLT_BLOCK_UUDT = 0x00000012
99 } T_PDU_FILTER;
100
101 typedef enum E_PDU_QUEUE_MODE
102 {
```

```
101     PDU_QUE_UNLIMITED = 0x00000000, /* In Unlimited Mode, the QueueSize is
102     ignored.*/
103     PDU_QUE_LIMITED = 0x00000001, /* When the ComLogicalLink's event queue
104     is full, no new items
105     are placed on the event queue.*/
106     PDU_QUE_CIRCULAR = 0x00000002 /* When the ComLogicalLink's event queue
107     is full (i.e. maximum
108     size has been reached), then the
109     oldest event item in the
110     queue is deleted so that the new event
111     item can then be
112     placed in the event queue.*/
113 } T_PDU_QUEUE_MODE;
114
115 typedef enum E_PDU_ERROR
116 {
117     PDU_STATUS_NOERROR = 0x00000000, /* No error for the function call */
118     PDU_ERR_FCT_FAILED = 0x00000001, /* Function call failed (generic
119     failure) */
120     PDU_ERR_RESERVED_1 = 0x00000010, /* Reserved by ISO 22900-2 */
121     PDU_ERR_COMM_PC_TO_VCI_FAILED = 0x00000011, /* Communication between
122     host and
123     MVCI protocol module
124     failed */
125     PDU_ERR_PDUAPI_NOT_CONSTRUCTED = 0x00000020, /* The D-PDU API has not
126     yet been constructed */
127     PDU_ERR_SHARING_VIOLATION = 0x00000021, /* A PDUdeconstruct was not called
128     before another PDUconstruct */
129     PDU_ERR_RESOURCE_BUSY = 0x00000030, /* the requested resource is
130     already in use.*/
131     PDU_ERR_RESOURCE_TABLE_CHANGED = 0x00000031, /* Not used by the D-PDU
132     API */
133     PDU_ERR_RESOURCE_ERROR = 0x00000032, /* Not used by the D-PDU API */
134     PDU_ERR_CLL_NOT_CONNECTED = 0x00000040, /* The ComLogicalLink cannot be
135     in the PDU_CLLST_OFFLINE state
136     to perform the requested
137     operation.*/
138     PDU_ERR_CLL_NOT_STARTED = 0x00000041, /* The ComLogicalLink must be in
139     the PDU_CLLST_COMM_STARTED
140     state to perform the requested
141     operation. */
142     PDU_ERR_INVALID_PARAMETERS = 0x00000050, /* One or more of the
143     parameters supplied in the function are
144     invalid. */
145     PDU_ERR_INVALID_HANDLE = 0x00000060, /* One or more of the handles
146     supplied in
147     the function are invalid. */
148     PDU_ERR_VALUE_NOT_SUPPORTED = 0x00000061, /* One of the option values
149     in PDUconstruct is invalid. */
150     PDU_ERR_ID_NOT_SUPPORTED = 0x00000062, /* IOCTL command id not
151     supported by the implementation of the
152     D-PDU API */
153     PDU_ERR_COMPARAM_NOT_SUPPORTED = 0x00000063, /* ComParam id not
154     supported by the implementation of the
155     D-PDU API */
156 }
```

```
135 PDU_ERR_COMPARAM_LOCKED = 0x00000064, /* Physical ComParam cannot be
changed because it is locked by
136                                     another ComLogicalLink. */
137 PDU_ERR_TX_QUEUE_FULL = 0x00000070, /* The ComLogicalLink's transmit
queue is full; the
138                                     ComPrimitive could not be queued
. */
139 PDU_ERR_EVENT_QUEUE_EMPTY = 0x00000071, /* No more event items are
available to be read from the
140                                     requested queue. */
141 PDU_ERR_VOLTAGE_NOT_SUPPORTED = 0x00000080, /* The voltage value
supplied in the IOCTL call is not
142                                     supported by the MVCI
protocol module. */
143 PDU_ERR_MUX_RSC_NOT_SUPPORTED = 0x00000081, /* The specified pin /
resource are not supported by the MVCI
144                                     protocol module for the
IOCTL call. */
145 PDU_ERR_CABLE_UNKNOWN = 0x00000082, /* The cable attached to the MVCI
protocol
146                                     module is of an unknown type. */
147 PDU_ERR_NO_CABLE_DETECTED = 0x00000083, /* No cable is detected by the
MVCI protocol module */
148 PDU_ERR_CLL_CONNECTED = 0x00000084, /* The ComLogicalLink is already in
the
149                                     PDU_CLLST_ONLINE state. */
150 PDU_ERR_TEMPPARAM_NOT_ALLOWED = 0x00000090, /* Physical ComParams
cannot be changed as a temporary
151                                     ComParam. */
152 PDU_ERR_RSC_LOCKED = 0x000000A0, /* The resource is already locked. */
153 PDU_ERR_RSC_LOCKED_BY_OTHER_CLL = 0x000000A1, /* The ComLogicalLink's
resource is currently locked by
154                                     another ComLogicalLink
. */
155 PDU_ERR_RSC_NOT_LOCKED = 0x000000A2, /* The resource is already in the
unlocked state. */
156 PDU_ERR_MODULE_NOT_CONNECTED = 0x000000A3, /* The module is not in the
PDU_MODST_READY state. */
157 PDU_ERR_API_SW_OUT_OF_DATE = 0x000000A4, /* The API software is older
than the
158                                     MVCI protocol module
Software*/
159 PDU_ERR_MODULE_FW_OUT_OF_DATE = 0x000000A5, /* The MVCI protocol module
software is older than the API
160                                     software. */
161 PDU_ERR_PIN_NOT_CONNECTED = 0x000000A6 /* The requested Pin is not
routed by supported cable */
162 } T_PDU_ERROR;
163
164 typedef enum E_PDU_ERR_EVT
165 {
166     PDU_ERR_EVT_NOERROR = 0x00000000, /* No Error. Event type only returned
on a PDUGetLastError if
167                                     there were no previous errors for
the requested handle */
```

```
168     PDU_ERR_EVT_FRAME_STRUCT = 0x00000100, /* CLL/CoP Error: The structure
169     of the received protocol frame
170                                     is incorrect (e.g. wrong
171     frame number, missing FC  $i\frac{1}{2}$ ). */
172     PDU_ERR_EVT_TX_ERROR = 0x00000101, /* CLL/CoP Error: Error encountered
173     during
174                                     transmit of a ComPrimitive PDU.
175                                     */
176     PDU_ERR_EVT_TESTER_PRESENT_ERROR = 0x00000102, /* CLL/CoP Error: Error
177     encountered in transmitting a Tester
178                                     Present message or in
179     receiving an expected response to a
180                                     Tester Present
181     message. */
182     PDU_ERR_EVT_RX_TIMEOUT = 0x00000103, /* CLL/CoP Error: Receive timer (e
183     .g. P2Max) expired with no
184                                     expected responses received
185     from the vehicle.*/
186     PDU_ERR_EVT_RX_ERROR = 0x00000104, /* CLL/CoP Error: Error encountered
187     in receiving a message
188                                     from the vehicle bus (e.g.
189     checksum error  $i\frac{1}{2}$ ). */
190     PDU_ERR_EVT_PROT_ERR = 0x00000105, /* CLL/CoP Error: Protocol error
191     encountered during handling
192                                     of a ComPrimitive (e.g. if the
193     protocol cannot handle the
194                                     length of a ComPrimitive).*/
195     PDU_ERR_EVT_LOST_COMM_TO_VCI = 0x00000106, /* Module Error:
196     Communication to a MVCI protocol module has
197                                     been lost.*/
198     PDU_ERR_EVT_VCI_HARDWARE_FAULT = 0x00000107, /* Module Error: The MVCI
199     protocol module has detected a
200                                     hardware error.*/
201     PDU_ERR_EVT_INIT_ERROR = 0x00000108, /* CLL/CoP Error: A failure
202     occurred during a protocol
203                                     initialization sequence. */
204     PDU_ERR_EVT_RSC_LOCKED = 0x00000109 /* CLL Error: A physical ComParam
205     was not set because of a
206                                     physical ComParam lock. */
207 } T_PDU_ERR_EVT;
208
209 typedef enum E_PDU_PC
210 {
211     PDU_PC_UNDEFINED = 0,
212     PDU_PC_TIMING = 1,
213     PDU_PC_INIT = 2,
214     PDU_PC_COM = 3,
215     PDU_PC_ERRHDL = 4,
216     PDU_PC_BUSTYPE = 5,
217     PDU_PC_UNIQUE_ID = 6,
218     PDU_PC_TESTER_PRESENT = 7
219 } T_PDU_PC;
220
221 typedef enum E_PDU_PT
222 {
223     PDU_PT_UNDEFINED = 0x00000000,
```

```
207     PDU_PT_UNUM8 = 0x00000101, /* Unsigned byte */
208     PDU_PT_SNUM8 = 0x00000102, /* Signed byte */
209     PDU_PT_UNUM16 = 0x00000103, /* Unsigned two bytes */
210     PDU_PT_SNUM16 = 0x00000104, /* Signed two bytes */
211     PDU_PT_UNUM32 = 0x00000105, /* Unsigned four bytes */
212     PDU_PT_SNUM32 = 0x00000106, /* Signed four bytes */
213     PDU_PT_BYTEFIELD = 0x00000107, /* Structure contains an array of UNUM8
length and actual length fields. See
214     ComParam BYTEFIELD
data type for the definition. */
215     PDU_PT_STRUCTFIELD = 0x00000108, /* Structure contains a void * pointer
to an array of
216     structures. The ComParamStructType
item determines the type
217     of structure to be typecasted onto
the void * pointer. This
218     structure contains a field for
maximum number of struct
219     entries and the actual number of
220     struct entries. See
ComParam STRUCTFIELD data type for
221     the definition. */
222     PDU_PT_LONGFIELD = 0x00000109 /* Structure contains an array of UNUM32
entries with a maximum
223     length and actual length fields. See
ComParam LONGFIELD Data
224     Type for the definition. */
225 } T_PDU_PT;
226
227 typedef enum E_PDU_CPST
228 {
229     PDU_CPST_SESSION_TIMING = 0x00000001, /* \see
PDU_PARAM_STRUCT_SESS_TIMING*/
230     PDU_CPST_ACCESS_TIMING = 0x00000002 /* \see
PDU_PARAM_STRUCT_ACCESS_TIMING*/
231 } T_PDU_CPST;
232
233 typedef struct
234 {
235     T_PDU_IT ItemType; /* See T_PDU_IT.*/
236 } PDU_ITEM;
237
238 typedef struct
239 {
240     T_PDU_IT ItemType; /* Value= one of the IOCTL constants from T_PDU_IT
(\ref
241     E_PDU_IT).*/
242     void* pData; /* Pointer to the specific IOCTL data structure.*/
243 } PDU_DATA_ITEM;
244
245 typedef struct
246 {
247     UNUM32 ProgVoltage_mv; /* Programming voltage [mV].*/
248     UNUM32 PinOnDLC; /* Pin number on Data Link Connector.*/
249 } PDU_IO_PROG_VOLTAGE_DATA;
```

```
250
251 typedef struct
252 {
253     UNUM32 DataSize; /* Number of bytes in the data array.*/
254     UNUM8* pData; /* Pointer to the data array.*/
255 } PDU_IO_BYTEARRAY_DATA;
256
257 typedef struct
258 {
259     T_PDU_FILTER
260     FilterType; /* type of filter being configured \see T_PDU_FILTER.*/
261     UNUM32 FilterNumber; /* Filter Number. Used to replace filters and stop
262                          filters.*/
263     UNUM32 FilterCompareSize; /* Number of bytes used out of each of the
264                               filter
265                               messages arrays Range 1-12.*/
266     UNUM8 FilterMaskMessage[12]; /* Mask message to be ANDED to each
267     incoming
268                                     message.*/
269     UNUM8 FilterPatternMessage[12]; /* Pattern message to be compared to
270     the
271                                     incoming message after the
272                                     FilterMaskMessage has been applied.
273     */
274 } PDU_IO_FILTER_DATA;
275
276 typedef struct
277 {
278     UNUM32
279     NumFilterEntries; /* Number of Filter entries in the filter list array.
280     */
281     PDU_IO_FILTER_DATA* pFilterData; /* Pointer to an array of filter data.
282     */
283 } PDU_IO_FILTER_LIST;
284
285 typedef struct
286 {
287     UNUM32 QueueSize; /* Maximum size of event queue. */
288     T_PDU_QUEUE_MODE QueueMode; /* Queue mode. see T_PDU_QUEUE_MODE event
289     queue
290     mode type values.*/
291 } PDU_IO_EVENT_QUEUE_PROPERTY_DATA;
292
293 typedef struct
294 {
295     UNUM32 hMod; /* Handle of a MVCI protocol module (IN parameter).*/
296     UNUM32 ResourceId; /* ID (IN parameter).*/
297     UNUM32 ResourceStatus; /* Resource Information Status (OUT Parameter).
298     */
299 } PDU_RSC_STATUS_DATA;
300
301 typedef struct
302 {
303     T_PDU_IT ItemType; /*! value=PDU_IT_RSC_STATUS (IN parameter).*/
304     UNUM32
```

```
297     NumEntries; /* number of entries in pResourceStatusData (IN Parameter)
298     */
298     PDU_RSC_STATUS_DATA* pResourceStatusData; /* array to contain resource
299     status (IN Parameter).*/
300 } PDU_RSC_STATUS_ITEM;
301
301 typedef struct
302 {
303     T_PDU_IT ItemType; /* value= PDU_IT_PARAM.*/
304     UNUM32 ComParamId; /* ComParam Id.*/
305     T_PDU_PT ComParamDataType; /* Defines the data type of the ComParam \
306     ref
307     T_PDU_PT; ComParam data type.*/
308     T_PDU_PC ComParamClass; /* ComParam Class type.*/
309     void* pComParamData; /* pointer to ComParam data of type
310     ComParamDataType.*/
311 } PDU_PARAM_ITEM;
312
312 typedef struct
313 {
314     UNUM32 ModuleTypeId; /* MVCI protocol moduleTypeId.*/
315     UNUM32 hMod; /* handle of MVCI protocol module assigned by D-PDU.*/
316     CHAR8* pVendorModuleName; /* Vendor specific information string for the
317     unique
318     module identification.*/
319     CHAR8* pVendorAdditionalInfo; /* Vendor specific additional information
320     string.*/
321     T_PDU_STATUS ModuleStatus; /* Status of MVCI protocol module detected
322     by D-PDU
323     API session.*/
324 } PDU_MODULE_DATA;
325
325 typedef struct
326 {
327     T_PDU_IT ItemType; /* value= PDU_IT_MODULE_ID */
328     UNUM32 NumEntries; /* number of entries written to the pModuleData
329     array */
330     PDU_MODULE_DATA* pModuleData; /* pointer to array containing module
331     types and
332     module handles */
333 } PDU_MODULE_ITEM;
334
334 typedef struct
335 {
336     UNUM32 hMod; /* Module handle*/
337     UNUM32 NumIds; /* number of resources that match PDU_RSC_DATA */
338     UNUM32* pResourceIdArray; /* pointer to a list of resource ids*/
339 } PDU_RSC_ID_ITEM_DATA;
340
340 typedef struct
341 {
342     T_PDU_IT ItemType; /* value = PDU_IT_RSC_ID (IN parameter)*/
343     UNUM32 NumModules; /* number of entries in pResourceIdDataArray. */
344     PDU_RSC_ID_ITEM_DATA
345     *pResourceIdDataArray; /* pointer to an array of resource Id Item Data
346     */
```



```
343 } PDU_RSC_ID_ITEM;
344
345 typedef struct
346 {
347     UNUM32 DLCPinNumber; /* Pin number on DLC */
348     UNUM32 DLCPinTypeId; /* Pin ID */
349 } PDU_PIN_DATA;
350
351 typedef struct
352 {
353     UNUM32 BusTypeId; /* Bus Type Id (IN parameter) */
354     UNUM32 ProtocolId; /* Protocol Id (IN parameter) */
355     UNUM32 NumPinData; /* Number of items in the following array */
356     PDU_PIN_DATA* pDLCPinData; /* Pointer to array of PDU_PIN_DATA
    structures*/
357 } PDU_RSC_DATA;
358
359 typedef struct
360 {
361     UNUM32 hMod; /* Handle of the MVCI protocol module with conflict.*/
362     UNUM32 ResourceId; /* Conflicting Resource ID.*/
363 } PDU_RSC_CONFLICT_DATA;
364
365 typedef struct
366 {
367     T_PDU_IT ItemType; /* value= PDU_IT_RSC_CONFLICT.*/
368     UNUM32 NumEntries; /* Number of entries written to pRscConflictData.*/
369     PDU_RSC_CONFLICT_DATA
370     *pRscConflictData; /* Pointer to array of PDU_RSC_CONFLICT_DATA.*/
371 } PDU_RSC_CONFLICT_ITEM;
372
373 typedef struct
374 {
375     UNUM32 UniqueRespIdentifier; /* filled out by application */
376     UNUM32 NumParamItems; /* number of ComParams for the Unique Identifier
    */
377     PDU_PARAM_ITEM
378     *pParams; /* pointer to array of ComParam items to uniquely define a
    ECU
379     response. The list is protocol specific */
380 } PDU_ECU_UNIQUE_RESP_DATA;
381
382 typedef struct
383 {
384     T_PDU_IT ItemType; /* Value= PDU_IT_UNIQUE_RESP_ID_TABLE */
385     UNUM32 NumEntries; /* Number of entries in the table.*/
386     PDU_ECU_UNIQUE_RESP_DATA* pUniqueData; /* Pointer to array of table
    entries for each ECU response.*/
387 } PDU_UNIQUE_RESP_ID_TABLE_ITEM;
388
389 typedef struct
390 {
391     T_PDU_IT ItemType; /* Value= PDU_IT_RESULT or PDU_IT_STATUS or
    PDU_IT_ERROR or
392     PDU_IT_INFO.*/
393     UNUM32
```

```
394     hCoP; /* If item is from a ComPrimitive then the hCoP contains the
        valid
395           ComPrimitive handle, else it contains PDU_HANDLE_UNDEF.*/
396     void* pCoPtag; /* ComPrimitive Tag. Should be ignored if
397                   hCoP=PDU_HANDLE_UNDEF.*/
398     UNUM32 Timestamp; /* Timestamp in microseconds.*/
399     void* pData; /* Points to the data for the specified Item Type.*/
400 } PDU_EVENT_ITEM;
401
402 typedef T_PDU_STATUS PDU_STATUS_DATA;
403
404 typedef struct
405 {
406     T_PDU_INFO InfoCode; /* Information code.*/
407     UNUM32 ExtraInfoData; /* Optional additional information.*/
408 } PDU_INFO_DATA;
409
410 typedef struct
411 {
412     T_PDU_ERR_EVT ErrorCodeId; /* Error code, binary information.*/
413     UNUM32 ExtraErrorInfoId; /* Optional additional error information, text
414                               translation via MDF file. Binary
415                               Information, 0
416                               indicates no additional error information.
417                               */
418 } PDU_ERROR_DATA;
419
420 typedef struct
421 {
422     UNUM32 NumFlagBytes; /* Number of bytes in pFlagData array.*/
423     UNUM8* pFlagData; /* Pointer to flag bytes used for TxFlag, RxFlag, and
424                       CllCreateFlag.*/
425 } PDU_FLAG_DATA;
426
427 typedef struct
428 {
429     UNUM32 NumHeaderBytes; /* Number of header bytes contained in
430                           pHeaderBytes
431                           array. */
432     UNUM32 NumFooterBytes; /* Number of footer bytes contained in
433                           pFooterBytes
434                           array. */
435     UNUM8* pHeaderBytes; /* Reference pointer to Response PDU Header bytes,
436                          NULL
437                          if NumHeaderBytes = 0.*/
438     UNUM8* pFooterBytes; /* Reference pointer to Response PDU Footer bytes,
439                          NULL
440                          if NumFooterBytes = 0.*/
441 } PDU_EXTRA_INFO;
442
443 typedef struct
444 {
445     PDU_FLAG_DATA RxFlag;
446     UNUM32 UniqueRespIdentifier;
447     UNUM32 AcceptanceId;
448     PDU_FLAG_DATA TimestampFlags;
```

```
443     UNUM32 TxMsgDoneTimestamp;
444     UNUM32 StartMsgTimestamp;
445     PDU_EXTRA_INFO* pExtraInfo;
446     UNUM32 NumDataBytes;
447     UNUM8* pDataBytes;
448 } PDU_RESULT_DATA;
449
450 typedef struct
451 {
452     UNUM32 MSCI_Part1StandardVersion;
453     UNUM32 MSCI_Part2StandardVersion;
454     UNUM32 HwSerialNumber;
455     CHAR8 HwName[64];
456     UNUM32 HwVersion;
457     UNUM32 HwDate;
458     UNUM32 HwInterface;
459     CHAR8 FwName[64];
460     UNUM32 FwVersion;
461     UNUM32 FwDate;
462     CHAR8 VendorName[64];
463     CHAR8 PDUApiSwName[64];
464     UNUM32 PDUApiSwVersion;
465     UNUM32 PDUApiSwDate;
466 } PDU_VERSION_DATA;
467
468 typedef struct
469 {
470     UNUM32 ResponseType; /* 0 = positive response; 1 = negative response.*/
471     UNUM32 AcceptanceId; /* ID assigned by application to be returned in
472                          PDU_RESULT_DATA, which indicates which expected
473                          response matched.*/
474     UNUM32 NumMaskPatternBytes; /* number of bytes in the Mask Data and
475                                Pattern
476                                Data.*/
477     UNUM8* pMaskData; /* Pointer to Mask Data. Bits set to a '1' are care
478                       bits,
479                       '0' are don't care bits.*/
480     UNUM8* pPatternData; /* Pointer to Pattern Data. Bytes to compare after
481                           the
482                           mask is applied.*/
483     UNUM32 NumUniqueRespIds; /* Number of items in the following array of
484                               unique
485                               response identifiers.*/
486     UNUM32* pUniqueRespIds; /* Array containing unique response identifiers
487                               . Only
488                               responses with a unique response identifier
489                               found
490                               in this array are considered, when trying to
491                               match
492                               them to this expected response. */
493 } PDU_EXP_RESP_DATA;
494
495 typedef struct
496 {
497     UNUM32 Time; /* Cycle time in ms for cyclic send operation or delay
498                 time for
```

```
491         PDU_COPT_DELAY ComPrimitive.*/
492     SNUM32 NumSendCycles; /* Number of send cycles to be performed.*/
493     SNUM32 NumReceiveCycles; /* Number of receive cycles to be performed.*/
494     UNUM32 TempParamUpdate; /* Temporary ComParam settings for the
ComPrimitive.*/
495     PDU_FLAG_DATA TxFlag; /* Transmit Flag used to indicate protocol
specific
496         elements for the ComPrimitive's execution..*/
497     UNUM32 NumPossibleExpectedResponses; /* number of entries in \ref
498         pExpectedResponseArray.*/
499     PDU_EXP_RESP_DATA
500     *pExpectedResponseArray; /* pointer to an array of expected responses.
*/
501 } PDU_COP_CTRL_DATA;
502
503 typedef struct
504 {
505     UNUM32 ParamMaxLen; /* Contains the maximum number of UNUM8 bytes the
ComParam
506         can contain in \ref pDataArray.*/
507     UNUM32 ParamActLen; /* Contains the actual number of UNUM8 bytes in
508         pDataArray..*/
509     UNUM8* pDataArray; /* Pointer to an array of \ref UNUM8 values.*/
510 } PDU_PARAM_BYTEFIELD_DATA;
511
512 typedef struct
513 {
514     T_PDU_CPST ComParamStructType; /* Type of ComParam Structure being used
.*/
515     UNUM32 ParamMaxEntries; /* Contains the maximum number of struct
entries in
516         \ref pStructArray.*/
517     UNUM32 ParamActEntries; /* Contains the maximum number of struct
entries the
518         ComParam can contain in \ref pStructArray.*/
519     void* pStructArray; /* Pointer to an array of structs (typecasted to
the \ref
520         ComParamStructType).*/
521 } PDU_PARAM_STRUCTFIELD_DATA;
522
523 typedef struct
524 {
525     UNUM16
526     session; /* Session Number, for the diagnostic session of ISO 15765-3.
*/
527     UNUM8 P2Max_high; /* Default P2Can_Server_max timing (1 ms resolution)
528         supported by the server for the activated
diagnostic
529         session. Used for ComParam CP_P2Max.*/
530     UNUM8 P2Max_low; /* Timing used for ComParam CP_P2Min (1 ms resolution)
.*/
531     UNUM8 P2Star_high; /* Enhanced (NRC 78 hex) P2Can_Server_max (10 ms
532         resolution) supported by the server for the
activated
533         diagnostic session. Used for ComParam CP_P2Star.
*/
```

```
534     UNUM8
535     P2Star_low; /* Timing (10 ms resolution) used for internal ECU use only
    */
536 } PDU_PARAM_STRUCT_SESS_TIMING;
537
538 typedef struct
539 {
540     UNUM8 P2Min; /* Minimum time (0,5 ms resolution) between tester request
    and
541                 ECU response(s). Used for ComParam CP_P2Min.*/
542     UNUM8 P2Max; /* Maximum time between tester request and ECU response(s)
    */
543     UNUM8 P3Min; /* Minimum time between end of ECU responses and start of
    new
544                 tester request.*/
545     UNUM8 P3Max; /* Maximum time between ECU responses and start of new
    tester
546                 request.*/
547     UNUM8 P4Min; /* Minimum inter byte time for tester request.*/
548     UNUM8 TimingSet; /* Set number allowing multiple sets of timing
    parameters.*/
549 } PDU_PARAM_STRUCT_ACCESS_TIMING;
550
551 typedef struct
552 {
553     UNUM32 ParamMaxLen; /* Contains the maximum number of UNUM32 entries
    the
554                         ComParam can contain in \ref pDataArray.*/
555     UNUM32 ParamActLen; /* Contains the current number of UNUM32 entries
    the
556                         ComParam can contain in \ref pDataArray.*/
557     UNUM32* pDataArray; /* Pointer to an array of UNUM32 values.*/
558 } PDU_PARAM_LONGFIELD_DATA;
559
560 typedef void (*CALLBACKFNC)(T_PDU_EVT_DATA eventType, UNUM32 hMod, UNUM32
    hCll, void* pCllTag, void* pAPITag);
561
562 EXTERNC T_PDU_ERROR PDUConstruct(CHAR8* pszOption, void* pAPITag);
563 EXTERNC T_PDU_ERROR PDUDestruct();
564 EXTERNC T_PDU_ERROR PDUModuleConnect(UNUM32 hMod);
565 EXTERNC T_PDU_ERROR PDUModuleDisconnect(UNUM32 hMod);
566 EXTERNC T_PDU_ERROR PDUGetModuleIds(PDU_MODULE_ITEM** pModuleIdList);
567 EXTERNC T_PDU_ERROR PDUGetResourceIds(UNUM32 hMod, PDU_RSC_DATA*
    pResourceIdData, PDU_RSC_ID_ITEM** pResourceIdList);
568 EXTERNC T_PDU_ERROR PDUGetResourceStatus(PDU_RSC_STATUS_ITEM*
    pResourceStatus);
569 EXTERNC T_PDU_ERROR PDUCreateComLogicalLink(UNUM32 hMod,
570     PDU_RSC_DATA* pRscData,
571     UNUM32 uiResourceId,
572     void* pCllTag,
573     UNUM32* pHcll,
574     PDU_FLAG_DATA* pCllCreateFlag);
575 EXTERNC T_PDU_ERROR PDUDestroyComLogicalLink(UNUM32 hMod, UNUM32 hc11);
576 EXTERNC T_PDU_ERROR PDURegisterEventCallback(UNUM32 hMod, UNUM32 hc11,
    CALLBACKFNC fnCB);
```

```
577 EXTERNC T_PDU_ERROR PDUGetComParam(UNUM32 hMod, UNUM32 hC11, UNUM32
    uiParamId, PDU_PARAM_ITEM** pParamItem);
578 EXTERNC T_PDU_ERROR PDUSetComParam(UNUM32 hMod, UNUM32 hC11, PDU_PARAM_ITEM
    * pParamItem);
579 EXTERNC T_PDU_ERROR PDUConnect(UNUM32 hMod, UNUM32 hC11);
580 EXTERNC T_PDU_ERROR PDUDisconnect(UNUM32 hMod, UNUM32 hC11);
581 EXTERNC T_PDU_ERROR PDUStartComPrimitive(UNUM32 hMod,
582     UNUM32 hC11,
583     T_PDU_COPT uiCoPType,
584     UNUM32 uiCoPDataSize,
585     UNUM8* pCoPData,
586     PDU_COP_CTRL_DATA* pCopCtrlData,
587     void* pCoPtag,
588     UNUM32* phCoP);
589 EXTERNC T_PDU_ERROR PDUCancelComPrimitive(UNUM32 hMod, UNUM32 hC11, UNUM32
    hCoP);
590 EXTERNC T_PDU_ERROR PDUGetEventItem(UNUM32 hMod, UNUM32 hC11,
    PDU_EVENT_ITEM** pEventItem);
591 EXTERNC T_PDU_ERROR PDUDestroyItem(PDU_ITEM* pItem);
592 EXTERNC T_PDU_ERROR PDUGetVersion(UNUM32 hMod, PDU_VERSION_DATA*
    pVersionData);
593 EXTERNC T_PDU_ERROR PDUGetLastError(UNUM32 hMod,
594     UNUM32 hC11,
595     T_PDU_ERR_EVT* pErrorCode,
596     UNUM32* phCoP,
597     UNUM32* pTimestamp,
598     UNUM32* pExtraErrorInfo);
599 EXTERNC T_PDU_ERROR PDUGetUniqueRespIdTable(UNUM32 hMod,
600     UNUM32 hC11,
601     PDU_UNIQUE_RESP_ID_TABLE_ITEM** pUniqueRespIdTable);
602 EXTERNC T_PDU_ERROR PDUSetUniqueRespIdTable(UNUM32 hMod,
603     UNUM32 hC11,
604     PDU_UNIQUE_RESP_ID_TABLE_ITEM* pUniqueRespIdTable);
605 EXTERNC T_PDU_ERROR
606 PDUIoctl(UNUM32 hMod, UNUM32 hC11, UNUM32 uiIoctlCommandId, PDU_DATA_ITEM*
    pInputData, PDU_DATA_ITEM** pOutputData);
607 EXTERNC T_PDU_ERROR PDUGetObjectId(UNUM32 uiPDUObjectType, CHAR8*
    pszShortName, UNUM32* pPDUObjectId);
608 EXTERNC T_PDU_ERROR PDUGetTimestamp(UNUM32 hMod, UNUM32* puiTimestamp);
609 EXTERNC T_PDU_ERROR PDUGetStatus(UNUM32 hMod,
610     UNUM32 hC11,
611     UNUM32 hCoP,
612     T_PDU_STATUS* puiStatusCode,
613     UNUM32* puiTimestamp,
614     UNUM32* puiExtraInfo);
615 EXTERNC T_PDU_ERROR PDUGetConflictingResources(UNUM32 uiResourceId,
616     PDU_MODULE_ITEM* pModuleList,
617     PDU_RSC_CONFLICT_ITEM** pConflictList);
618 EXTERNC T_PDU_ERROR PDUlockResource(UNUM32 hMod, UNUM32 hC11, UNUM32
    uiLockMask);
619 EXTERNC T_PDU_ERROR PDUUnlockResource(UNUM32 hMod, UNUM32 hC11, UNUM32
    uiLockMask);
620
621
622 using BYTEFIELD_PTR = std::shared_ptr<const PDU_PARAM_BYTEFIELD_DATA>;
623 using STRUCTFIELD_PTR = std::shared_ptr<const PDU_PARAM_STRUCTFIELD_DATA>;
```

```
624 using LONGFIELD_PTR = std::shared_ptr<const PDU_PARAM_LONGFIELD_DATA>;
625
626 using ComParamData
627     = std::variant<UNUM8, SNUM8, UNUM16, SNUM16, UNUM32, SNUM32,
        BYTEFIELD_PTR, STRUCTFIELD_PTR, LONGFIELD_PTR>;
628
629 /// @brief ComParamDictionary
630 class ComParamDictionary
631 {
632 public:
633     virtual ~ComParamDictionary() = default;
634
635     ///
636     /// @brief Get the ComParam based on its name.
637     ///
638     /// @param name ComParam name (e.g, CP_P2Max)
639     /// @return ComParam value
640     ///
641     virtual ComParamData getComParamByName(ara::core::StringView name)
        const = 0;
642 };
643
644 /// @brief Communication Parameter Buffer
645 class ComParamBuffer : public ComParamDictionary
646 {
647 public:
648     ///
649     /// @brief Set ComParam based on the given PDU_PARAM_ITEM structure.
650     ///
651     /// @param param reference to PDU_PARAM_ITEM structure to be set
652     /// @return true if the ComParam is successfully set.
653     /// @return false if the ComParam Id is invalid or the paramType is
        different
654     ///         than UNUM32
655     ///
656     virtual bool setComParam(const PDU_PARAM_ITEM& param) = 0;
657
658     ///
659     /// @brief Get a ComParam based on its Id.
660     ///
661     /// @param paramId ComParam Id
662     /// @return pointer to the filled PDU_PARAM_ITEM structure or Null
        pointer if
663     ///         the ParamId is invalid.
664     ///
665     virtual std::shared_ptr<PDU_PARAM_ITEM> getComParam(UNUM32 paramId)
        const = 0;
666 };
667
668 /// @brief Communication Parameter Buffer implementation
669 class ComParamBufferImpl final : public ComParamBuffer
670 {
671 public:
672     /// @copybrief ComParamBuffer::setComParam()
673     /// @copydetails ComParamBuffer::setComParam()
674     bool setComParam(const PDU_PARAM_ITEM& param) override;
```

```
675
676     /// @copybrief ComParamBuffer::getComParam()
677     /// @copydetails ComParamBuffer::setComParam()
678     std::shared_ptr<PDU_PARAM_ITEM> getComParam(UNUM32 paramId) const
        override;
679
680     /// @copybrief ComParamDictionary::getComParamByName
681     /// @copydetails ComParamDictionary::getComParamByName
682     ComParamData getComParamByName(ara::core::StringView name) const
        override;
683
684     ///
685     /// @brief Add a ComParam to the ComParams list (@params_)
686     ///
687     /// @param id ComParam Id
688     /// @param name ComParam name
689     /// @param paramType ComParam Type
690     /// @param paramClass ComParam Class
691     /// @param data ComParam value
692     /// @return true if the ComParam is successfully added.
693     ///         false if the ComParam Id/name is invalid or the paramType
        is
694     ///         different than UNUM32
695     ///
696     bool addComParam(UNUM32 id,
697                     const ara::core::String& name,
698                     T_PDU_PT paramType,
699                     T_PDU_PC paramClass,
700                     const void* data);
701
702     ///
703     /// @brief Get the number of the ComParams.
704     ///
705     /// @return number of the ComParams
706     std::size_t size() const;
707
708     ///
709     /// @brief Add a ComParam to the URID table (@paramBuffer_)
710     ///
711     /// @param id ComParam Id
712     /// @param name ComParam name
713     /// @param paramType ComParam Type
714     /// @param paramClass ComParam Class
715     /// @param data ComParam value
716     /// @return true if the ComParam is successfully added.
717     ///         false if the parameter class is different than
        PDU_PC_UNIQUE_ID
718     ///
719     ara::core::String getComParamNameById(UNUM32 paramId) const;
720
721     /// @brief ComParam Key (Comparam Id and name)
722     struct ParamKey
723     {
724         UNUM32 id;
725         ara::core::String name;
726     };
```



```
727
728 // @brief ComParam Type and Class
729 struct ComParameter
730 {
731     const T_PDU_PT paramType;
732     const T_PDU_PC paramClass;
733     ComParamData data;
734 };
735
736 // @brief ComParamKey compare
737 struct cmpParamKey
738 {
739     bool operator()(const ParamKey& a, const ParamKey& b) const
740     {
741         if (a.id == b.id || a.name == b.name)
742             return false;
743         return (a.id < b.id) || (a.name < b.name);
744     }
745 };
746
747 using const_iterator = ara::core::Map<ParamKey, ComParameter,
748 cmpParamKey>::const_iterator;
749 const_iterator cbegin() const
750 {
751     return params_.cbegin();
752 }
753 const_iterator cend() const
754 {
755     return params_.cend();
756 }
757
758 private:
759 // @brief List of ComParams
760 ara::core::Map<ParamKey, ComParameter, cmpParamKey> params_;
761 };
762
763 // @brief URID Table Entry
764 class URIDTableEntry : public ComParamDictionary
765 {
766 public:
767     // @brief Unique Response Identifier
768     virtual UNUM32 uniqueRespIdentifier() const = 0;
769 };
770
771 // @brief Unique Response Identifier Table
772 class URIDTable
773 {
774 public:
775     virtual ~URIDTable() = default;
776
777     //
778     // @brief Check if there is an entry in the URID table.
779     //
780     // @return true if there is no entry in the URID table.
781     // @return false if there is one or more entries in the URID table.
```

```
782     ///  
783     virtual bool empty() const = 0;  
784     virtual const URIDTableEntry& operator[](std::size_t pos) const = 0;  
785  
786     ///  
787     ///  
788     ///  
789     ///  
790     ///  
791     virtual std::shared_ptr<PDU_UNIQUE_RESP_ID_TABLE_ITEM> getTableItem()  
const = 0;  
792  
793     ///  
794     ///  
795     ///  
796     ///  
797     ///  
798     ///  
799     ///  
800     virtual bool set(const PDU_UNIQUE_RESP_ID_TABLE_ITEM&) = 0;  
801 };  
802 ///  
803 class URIDTableEntryImpl final : public URIDTableEntry  
804 {  
805 public:  
806     explicit URIDTableEntryImpl(UNUM32 uniqueRespIdentifier = PDU_ID_UNDEF)  
807     ;  
808     ///  
809     UNUM32 uniqueRespIdentifier() const override;  
810  
811     ///  
812     ///  
813     ComParamData getComParamByName(ara::core::StringView name) const  
override;  
814  
815     ///  
816     ///  
817     ///  
818     ///  
819     ///  
820     ///  
821     ///  
822     std::shared_ptr<PDU_PARAM_ITEM> getComParam(UNUM32 paramId) const;  
823  
824     ///  
825     ///  
826     ///  
827     ///  
828     ///  
829     ///  
830     ///  
831     ///  
831     ///  

```

```
832     /// @return true if the ComParam is successfully added.
833     ///         false if the parameter class is different than
PDU_PC_UNIQUE_ID
834     ///
835     bool addComParam(UNUM32 id,
836                     const ara::core::String& name,
837                     T_PDU_PT paramType,
838                     T_PDU_PC paramClass,
839                     const void* data);
840
841     std::size_t size() const;
842     ara::core::String getComParamNameById(UNUM32 paramId) const;
843
844     using const_iterator = ComParamBufferImpl::const_iterator;
845     const_iterator cbegin() const
846     {
847         return paramBuffer_->cbegin();
848     }
849
850     const_iterator cend() const
851     {
852         return paramBuffer_->cend();
853     }
854
855 private:
856     /// @brief Unique Response Identifier
857     UNUM32 uniqueRespIdentifier_;
858
859     /// @brief URID table
860     std::unique_ptr<ComParamBufferImpl> paramBuffer_;
861 };
862
863 /// @brief URID Table implementation
864 class URIDTableImpl final : public URIDTable
865 {
866 public:
867     /// @copybrief URIDTable::empty()
868     /// @copydetails URIDTable::empty()
869     bool empty() const override;
870     const URIDTableEntry& operator[](std::size_t pos) const override;
871
872     /// @copybrief URIDTable::getTableItem()
873     /// @copydetails URIDTable::getTableItem()
874     std::shared_ptr<PDU_UNIQUE_RESP_ID_TABLE_ITEM> getTableItem() const
875     override;
876     bool set(const PDU_UNIQUE_RESP_ID_TABLE_ITEM& tableItem) override;
877
878     /// @copybrief URIDTable::URIDTableImpl()
879     /// @copydetails URIDTable::URIDTableImpl()
880     explicit URIDTableImpl(URIDTableEntryImpl&& entry);
881
882 private:
883     /// @brief URID table entries
884     ara::core::Vector<URIDTableEntryImpl> entries_;
885 };
```



```
940     /// @param hMod Handle of the MVCI protocol module to be disconnected.
941     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
942     ///         constructed,
943     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module Handle
is
944     ///         invalid.
945     ///         PDU_STATUS_NOERROR if the module is successfully
disconnected.
946     ///
947     T_PDU_ERROR PDUModuleDisconnect (UNUM32 hMod);
948
949     ///
950     /// @brief Get module type Id, module handle information, vendor-
specific
951     ///         string information and module status. Allocate
PDU_MODULE_ITEM
952     ///         structure and fill the call-by-reference variable
pModuleIdList
953     /// @param pModuleIdList Pointer for storing the pointer to Module Type
Ids
954     ///         and the Module handles for all the modules that are
connected.
955     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
956     ///         constructed,
957     ///         PDU_ERR_INVALID_PARAMETERS if the pModuleIdList parameter
is
958     ///         NULL.
959     ///         PDU_STATUS_NOERROR if the function is successfully called.
960     ///
961     T_PDU_ERROR PDUGetModuleIds (PDU_MODULE_ITEM** pModuleIdList);
962
963     ///
964     /// @brief Destroy the given item.
965     ///
966     ///         Free memory reserved by the D-PDU API for the given item.
967     /// @param pItem Pointer to the item to be destroyed.
968     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
969     ///         constructed,
970     ///         PDU_ERR_INVALID_PARAMETERS if the pItem is invalid or NULL,
971     ///         PDU_STATUS_NOERROR if the item type is successfully
destroyed.
972     ///
973     T_PDU_ERROR PDUDestroyItem (PDU_ITEM* pItem);
974
975     ///
976     /// @brief Create a ComLogicalLink for the given resource Id.
977     ///
978     ///         Stores the ComLogicalLink handle in pCll parameter.
979     /// @param hMod Handle of the MVCI protocol module.
980     /// @param pRscData Not used.
981     /// @param resourceId Resource Id
982     /// @param pCllTag Not used.
983     /// @param pCll Call-by-reference for storage of the ComLogicalLink
handle.
984     /// @param pCllCreateFlag Not used.
985     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
```

```
986     ///  
987     ///  
is  
988     ///  
989     ///  
for  
990     ///  
991     ///  
created.  
992     ///  
993     T_PDU_ERROR PDUCreateComLogicalLink(UNUM32 hMod,  
994         PDU_RSC_DATA* pRscData,  
995         UNUM32 resourceId,  
996         void* pCllTag,  
997         UNUM32* pHcll,  
998         PDU_FLAG_DATA* pCllCreateFlag);  
999  
1000    ///  
1001    ///  
1002    ///  
1003    ///  
1004    ///  
1005    ///  
1006    ///  
1007    ///  
1008    ///  
1009    ///  
1010    ///  
1011    ///  
has not  
1012    ///  
1013    ///  
1014    ///  
1015    ///  
1016    T_PDU_ERROR PDUDestroyComLogicalLink(UNUM32 hMod, UNUM32 hCll);  
1017  
1018    ///  
1019    ///  
1020    ///  
1021    ///  
1022    ///  
1023    ///  
1024    ///  
1025    ///  
1026    ///  
constructed,  
1027    ///  
1028    ///  
1029    ///  
1030    ///  
the  
1031    ///  
1032    ///  
1033    ///  
connected.
```

```

1034    ///
1035    T_PDU_ERROR PDUConnect(UNUM32 hMod, UNUM32 hCll);
1036
1037    ///
1038    /// @brief Disconnect the ComLogicalLink from the vehicle bus.
1039    ///
1040    /// @param hMod Handle of MVCI protocol module.
1041    /// @param hCll Handle of the ComLogicalLink to be disconnected.
1042    /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
1043    constructed,
1044    ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
1045    ',
1046    ///         PDU_ERR_CLL_NOT_CONNECTED if the ComLogicalLink is not
1047    connected,
1048    ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1049    ComLogicalLink handles are invalid,
1050    ///         PDU_STATUS_NOERROR if the ComLogicalLink is successfully
1051    disconnected.
1052    ///
1053    T_PDU_ERROR PDUDisconnect(UNUM32 hMod, UNUM32 hCll);
1054
1055    ///
1056    /// @brief Register a callback function for event notification.
1057    ///
1058    ///         Add the callback function pointer to the proper
1059    ComLogicalLink
1060    ///         object.
1061    /// @param hMod Handle of MVCI protocol module.
1062    /// @param hCll Handle of the ComLogicalLink.
1063    /// @param fnCB Reference of callback function to be used for event
1064    notification.
1065    /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
1066    constructed,
1067    ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
1068    ',
1069    ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1070    ComLogicalLink handles are invalid,
1071    ///         PDU_ERR_FCT_FAILED if the fnCB pointer is NULL.
1072    ///         PDU_STATUS_NOERROR if the callback function is successfully
1073    registered.
1074    ///
1075    T_PDU_ERROR PDURegisterEventCallback(UNUM32 hMod, UNUM32 hCll,
1076    CALLBACKFNC fnCB);
1077
1078    ///
1079    /// @brief Get the event item data (PDU_EVENT_ITEM) for the given event
1080    source.
1081    ///
1082    ///         Allocate memory for PDU_EVENT_ITEM and fill out the event
1083    item
1084    ///         information.
1085    /// @param hMod Handle of MVCI protocol module.
1086    /// @param hCll Handle of the ComLogicalLink.
1087    /// @param pEventItem Call-by-reference place for storing the pointer
1088    to the
1089    /// event item corresponding to the given event, hMod and hCll.
```

```
1081     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
1082     constructed,
1083     ///         PDU_ERR_INVALID_PARAMETERS if pEventItem pointer is NULL.
1084     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1085     ///         ComLogicalLink handles are invalid,
1086     ///         PDU_ERR_EVENT_QUEUE_EMPTY if no more event items are
1087     available,
1088     ///         PDU_STATUS_NOERROR if the event item is successfully
1089     retrieved.
1090     ///
1091     T_PDU_ERROR PDUGetEventItem(UNUM32 hMod, UNUM32 hCll, PDU_EVENT_ITEM**
1092     pEventItem);
1093
1094     ///
1095     /// @brief Creates a ComPrimitive for sending/receiving data and place
1096     it in
1097     ///         the CoP Queue.
1098     ///
1099     /// @param hMod Handle of MVCI protocol module.
1100     /// @param hCll Handle of the ComLogicalLink.
1101     /// @param uiCoPType Type of the ComPrimitive to be started.
1102     /// @param uiCoPDataSize Size of the ComPrimitive data.
1103     /// @param pCoPData Reference of the buffer holding the data.
1104     /// @param pCopCtrlData Pointer to the control data structure for the
1105     ComPrimitive.
1106     /// @param pCoPTag Not used.
1107     /// @param phCoP Call-by-reference place for storage of ComPrimitive
1108     handle
1109     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
1110     constructed,
1111     ///         PDU_ERR_INVALID_PARAMETERS if pCoPData or pCopCtrlData
1112     pointers
1113     ///         are NULL.
1114     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1115     ///         ComLogicalLink handles are invalid,
1116     ///         PDU_ERR_FCT_FAILED if uiCoPType is different than
1117     ///         PDU_COPT_SENDRECV (the current D-PDU API implementation
1118     support
1119     ///         only the Send-Receive type)
1120     ///         PDU_STATUS_NOERROR if the ComPrimitive is successfully
1121     started.
1122     ///
1123     T_PDU_ERROR PDUStartComPrimitive(UNUM32 hMod,
1124     UNUM32 hCll,
1125     T_PDU_COPT uiCoPType,
1126     UNUM32 uiCoPDataSize,
1127     UNUM8* pCoPData,
1128     PDU_COP_CTRL_DATA* pCopCtrlData,
1129     void* pCoPTag,
1130     UNUM32* phCoP);
1131
1132     ///
1133     /// @brief Get a ComParam from the ComParams buffer for the given
1134     module and
1135     /// ComLogicalLink handles.
1136     ///
```



```
1126     ///          Allocate memory for the PDU_PARAM_ITEM results and fill out
1127     the
1128     ///          ComParam information in it.
1129     /// @param hMod Handle of MVCI protocol module.
1130     /// @param hCll Hanlde of the ComLogicalLink.
1131     /// @param uiParamId ID value of the ComParam to be requested (see MDF
1132     to get
1133     ///          the list of ComParam IDs).
1134     /// @param pParamItem Call-by-reference place for storing the ComParam
1135     item.
1136     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
1137     constructed,
1138     ///          PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
1139     ,
1140     ///          PDU_ERR_INVALID_PARAMETERS if uiParamId is invalid or
1141     pParamItem
1142     ///          pointer is NULL,
1143     ///          PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1144     ///          ComLogicalLink handles are invalid,
1145     ///          PDU_STATUS_NOERROR if the requested ComParam is
1146     successfully
1147     ///          retrieved.
1148     ///
1149     T_PDU_ERROR PDUGetComParam(UNUM32 hMod, UNUM32 hCll, UNUM32 uiParamId,
1150     PDU_PARAM_ITEM** pParamItem);
1151
1152     ///
1153     /// @brief Set a ComParam in the ComParams buffer for the given module
1154     and
1155     ///          ComLogicalLink handles.
1156     ///
1157     /// @param hMod Handle of MVCI protocol module.
1158     /// @param hCll Hanlde of the ComLogicalLink.
1159     /// @param pParamItem ComParam item structure with the ComParam element
1160     to be
1161     ///          set.
1162     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
1163     constructed,
1164     ///          PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
1165     ,
1166     ///          PDU_ERR_INVALID_PARAMETERS if pParamItem pointer is NULL or
1167     the
1168     ///          ComParam ID is invalid,
1169     ///          PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1170     ///          ComLogicalLink handles are invalid,
1171     ///          PDU_ERR_FCT_FAILED if the ComLogicalLink is in "online"
1172     state,
1173     ///          PDU_STATUS_NOERROR if the given ComParam is successfully
1174     set.
1175     ///
1176     T_PDU_ERROR PDUSetComParam(UNUM32 hMod, UNUM32 hCll, PDU_PARAM_ITEM*
1177     pParamItem);
1178
1179     ///
1180     /// @brief Get information of all unique response identifiers
1181     configured for
```

```
1165     ///         the ComLogicalLink.
1166     ///
1167     /// @param hMod Handle of MCCI protocol module.
1168     /// @param hCll Handle of the ComLogicalLink.
1169     /// @param pUniqueRespIdTable Call-by-reference place for storing the
Unique
1170     ///         Response ID Table for the given CLL.
1171     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
constructed,
1172     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
',
1173     ///         PDU_ERR_INVALID_PARAMETERS if pUniqueRespIdTable pointer is
NULL
1174     ///         or the ComParam ID is invalid,
1175     ///         PDU_ERR_INVALID_HANDLE if the MCCI protocol module or
1176     ///         ComLogicalLink handles are invalid,
1177     ///         PDU_STATUS_NOERROR if the URID table is successfully
retrieved.
1178     ///
1179     T_PDU_ERROR
1180     PDUGetUniqueRespIdTable(UNUM32 hMod, UNUM32 hCll,
PDU_UNIQUE_RESP_ID_TABLE_ITEM** pUniqueRespIdTable);
1181
1182     ///
1183     /// @brief Set Unique Response Id Table for a ComLogicalLink.
1184     ///
1185     /// @param hMod Handle of MCCI protocol module.
1186     /// @param hCll Handle of the ComLogicalLink.
1187     /// @param pUniqueRespIdTable Call-by-reference witch contains the URID
table
1188     ///         for the given ComLogicalLink.
1189     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
constructed,
1190     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
',
1191     ///         PDU_ERR_INVALID_PARAMETERS if pUniqueRespIdTable pointer is
NULL
1192     ///         or the ComParam ID is invalid,
1193     ///         PDU_ERR_INVALID_HANDLE if the MCCI protocol module or
1194     ///         ComLogicalLink handles are invalid,
1195     ///         PDU_ERR_FCT_FAILED if the ComLogicalLink is in "online"
state.
1196     ///         PDU_STATUS_NOERROR if the URID table is successfully
retrieved.
1197     ///
1198     T_PDU_ERROR
1199     PDUSetUniqueRespIdTable(UNUM32 hMod, UNUM32 hCll,
PDU_UNIQUE_RESP_ID_TABLE_ITEM* pUniqueRespIdTable);
1200
1201     ~PduApi() = default;
1202
1203 private:
1204     enum class PduApiState
1205     {
1206         kUnconstructed = 0,
1207         kConstructed
```

```
1208     };
1209
1210     using PduDataType = std::variant<std::shared_ptr<PDU_EVENT_ITEM>,
1211         std::shared_ptr<PDU_PARAM_ITEM>,
1212         std::shared_ptr<PDU_RSC_DATA>,
1213         std::shared_ptr<PDU_MODULE_ITEM>,
1214         std::shared_ptr<PDU_RSC_ID_ITEM>,
1215         std::shared_ptr<PDU_UNIQUE_RESP_ID_TABLE_ITEM>>;
1216
1217     PduApi();
1218
1219     /// @brief Allocate PDU_MODULE_ITEM structure and fill the call-by-
1220     reference
1221     /// variable pModuleIdList
1222     ///
1223     /// @return the filled PDU_MODULE_ITEM structure
1224     std::shared_ptr<PDU_MODULE_ITEM> createModuleItem();
1225
1226     /// @brief D-PDU API states (constructed, Not constructed)
1227     PduApiState pduapiStatus_{PduApiState::kUnconstructed};
1228
1229     /// @brief list of the available MVCI protocol modules.
1230     ara::core::Map<UNUM32, Module> availableModules_;
1231
1232     /// @brief D-PDU API data, events and param items
1233     std::list<PduDataType> pduData_;
1234 };
```

## G Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

### G.1 Traceable item history of this document according to AUTOSAR Release R23-11

#### G.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_VUCM_00136]	Definition of Application Error Domain of functional cluster VUCM
[SWS_VUCM_00177]	Definition of ImplementationDataType SwNameVersionVectorType
[SWS_VUCM_00178]	Definition of Port VehiclePackageManagement provided by functional cluster VUCM
[SWS_VUCM_00179]	Definition of Port VehicleStateManagerInterface provided by functional cluster VUCM
[SWS_VUCM_00180]	Definition of Port VehicleDriverApplicationInterface provided by functional cluster VUCM
[SWS_VUCM_00181]	Definition of ServiceInterface VehiclePackageManagement
[SWS_VUCM_00182]	Definition of ServiceInterface VehicleDriverApplicationInterface
[SWS_VUCM_00183]	Definition of ServiceInterface VehicleStateManagerInterface
[SWS_VUCM_00210]	Transferring of software packages on <code>kProcessing</code> state
[SWS_VUCM_00251]	Definition of ImplementationDataType CampaignHistoryType
[SWS_VUCM_00252]	Definition of ImplementationDataType CampaignResultType
[SWS_VUCM_00253]	Definition of ImplementationDataType UCMStepErrorType
[SWS_VUCM_00254]	Definition of ImplementationDataType UCMHistoryVectorType
[SWS_VUCM_00255]	Definition of ImplementationDataType SoftwarePackageStepType
[SWS_VUCM_00256]	Definition of ImplementationDataType UCMMasterResolutionType
[SWS_VUCM_00268]	Definition of ImplementationDataType SwPackageDescType
[SWS_VUCM_00269]	Definition of ImplementationDataType SwPackageDescVectorType
[SWS_VUCM_00290]	Definition of ImplementationDataType UCMHistoryType
[SWS_VUCM_00291]	Definition of ImplementationDataType UCMStepErrorVectorType
[SWS_VUCM_00296]	Definition of ImplementationDataType CampaignHistoryVectorType
[SWS_VUCM_00297]	Retry Strategy for ServiceBusy
[SWS_VUCM_00298]	Retry Strategy for UpdateSessionRejected
[SWS_VUCM_00304]	Definition of ImplementationDataType UCMMasterResolutionVectorType
[SWS_VUCM_00307]	Definition of ImplementationDataType VehicleUCMInfo
[SWS_VUCM_00308]	Definition of ImplementationDataType UCMLIdentifiersAndVersionsType





Number	Heading
[SWS_VUCM_01003]	V-UCM checks states of UCMS
[SWS_VUCM_01005]	V-UCM is discovering UCMS in vehicle
[SWS_VUCM_01011]	TransferVehiclePackage InsufficientMemory
[SWS_VUCM_01013]	Too big block size received by V-UCM
[SWS_VUCM_01014]	Packages transferring sequence
[SWS_VUCM_01015]	Invalid Vehicle Package manifest
[SWS_VUCM_01016]	Invalid Package Manifest
[SWS_VUCM_01017]	RequestedPackage field
[SWS_VUCM_01018]	TransferVehiclePackage kBusyWithCampaign
[SWS_VUCM_01019]	V-UCM initialization
[SWS_VUCM_01020]	Retry Strategy for BlockInconsistent
[SWS_VUCM_01021]	Definition of Port PackageManagement required by functional cluster VUCM
[SWS_VUCM_01101]	Provide information of installed Software Clusters in vehicle
[SWS_VUCM_01103]	Inform Backend of needed Software Packages for an update
[SWS_VUCM_01105]	Interaction of V-UCM with Vehicle Driver
[SWS_VUCM_01109]	V-UCM provides a safety interface
[SWS_VUCM_01114]	Definition of ImplementationDataType SafetyConditionType
[SWS_VUCM_01117]	V-UCM VehicleConditionCollection field
[SWS_VUCM_01118]	V-UCM waiting for vehicle driver approval
[SWS_VUCM_01119]	Report information of Software Packages
[SWS_VUCM_01120]	Provide Software Packages general information
[SWS_VUCM_01122]	Supported physical layers by D-PDU API implementation
[SWS_VUCM_01123]	Supported application layers by D-PDU API implementation
[SWS_VUCM_01124]	Supported protocols by D-PDU API implementation
[SWS_VUCM_01131]	PDUIoCtl(PDU_IOCTL_RESET)
[SWS_VUCM_01132]	PDUIoCtl(PDU_IOCTL_START_MSG_FILTER), PDUIoCtl(PDU_IOCTL_CLEAR_MSG_FILTER), PDUIoCtl(PDU_IOCTL_STOP_MSG_FILTER)
[SWS_VUCM_01135]	Get Software Clusters descriptions from a vehicle
[SWS_VUCM_01138]	Definition of ImplementationDataType SafetyStateType
[SWS_VUCM_01139]	V-UCM configured behavior when vehicle safety is not met
[SWS_VUCM_01177]	Definition of ImplementationDataType CampaignStateType
[SWS_VUCM_01178]	Definition of ImplementationDataType TransferStateType
[SWS_VUCM_01201]	Sequential orchestration of campaigns
[SWS_VUCM_01203]	CampaignState field
[SWS_VUCM_01204]	Initial state
[SWS_VUCM_01205]	V-UCM internal state persistency
[SWS_VUCM_01207]	Trigger on kSoftwarePackage_Transferring state
[SWS_VUCM_01209]	Trigger on kProcessing state





Number	Heading
[SWS_VUCM_01212]	Trigger on <code>kActivating</code> state
[SWS_VUCM_01214]	Final action on <code>kVehicleChecking</code> state
[SWS_VUCM_01215]	Trigger on <code>kCancelling</code> state
[SWS_VUCM_01216]	Final action on <code>kCancelling</code> state
[SWS_VUCM_01217]	Monitoring of UCMs
[SWS_VUCM_01218]	Transition from <code>kIdle</code> state to <code>kSyncing</code> state
[SWS_VUCM_01219]	Transition from <code>kSyncing</code> state to <code>kIdle</code> state
[SWS_VUCM_01220]	Transition from <code>kIdle</code> state to <code>kVehiclePackageTransferring</code> and <code>kTransferring</code> states
[SWS_VUCM_01221]	Transition from <code>kVehiclePackageTransferring</code> state and <code>kTransferring</code> state to <code>kCancelling</code> state
[SWS_VUCM_01222]	Transition from <code>kVehiclePackageTransferring</code> state to <code>kSoftwarePackage_Transferring</code> state
[SWS_VUCM_01227]	Transition from <code>kSoftwarePackage_Transferring</code> state and <code>kTransferring</code> state to <code>kCancelling</code> state
[SWS_VUCM_01228]	Transition from <code>kSoftwarePackage_Transferring</code> state and <code>kTransferring</code> state to <code>kProcessing</code> state and <code>kUpdating</code> state
[SWS_VUCM_01229]	SafetyConditions while processing stream
[SWS_VUCM_01234]	Transition from <code>kProcessing</code> state to <code>kActivating</code> state
[SWS_VUCM_01236]	Transition from <code>kProcessing</code> state and <code>kUpdating</code> state to <code>kCancelling</code> state
[SWS_VUCM_01239]	Transition from <code>kActivating</code> state and <code>kUpdating</code> state to <code>kCancelling</code> state
[SWS_VUCM_01240]	Transition from <code>kActivating</code> state to <code>kVehicleChecking</code> state
[SWS_VUCM_01241]	Transition from <code>kVehicleChecking</code> state and <code>kUpdating</code> state to <code>kCancelling</code> state
[SWS_VUCM_01242]	Transition from <code>kVehicleChecking</code> state and <code>kUpdating</code> state to <code>kIdle</code> state
[SWS_VUCM_01243]	Transition from <code>kCancelling</code> state to <code>kIdle</code> state
[SWS_VUCM_01244]	Cancellation of an update campaign shall be possible
[SWS_VUCM_01246]	Unreachable UCM during update campaign
[SWS_VUCM_01247]	Method to read History Report
[SWS_VUCM_01248]	Content of History Report
[SWS_VUCM_01265]	TransferState field
[SWS_VUCM_01266]	Subordinate Not Available On The Network
[SWS_VUCM_01267]	Vehicle State Manager Communication Error
[SWS_VUCM_01268]	Vehicle Driver Interface Communication Error
[SWS_VUCM_01269]	Campaign cancellation history
[SWS_VUCM_01270]	New campaign disabling
[SWS_VUCM_01271]	New campaign enabling
[SWS_VUCM_01272]	<code>VehicleCheck</code> call not permitted





Number	Heading
[SWS_VUCM_01273]	CancelCampaign kCancelFailed error
[SWS_VUCM_01274]	CancelCampaign kOperationNotPermitted error
[SWS_VUCM_01275]	Safety conditions during activation
[SWS_VUCM_01276]	Transition from kRollingBackFailed state to kIdle state
[SWS_VUCM_01277]	Transition from kCancelling state to kRollingBackFailed state
[SWS_VUCM_01278]	V-UCM behaviour in case a safety condition is not supported by Vehicle State Manager
[SWS_VUCM_01279]	Keep history of Driver notification during campaign
[SWS_VUCM_01280]	Maximum campaign duration
[SWS_VUCM_01301]	Vehicle Package authentication
[SWS_VUCM_01302]	Vehicle Package authentication failure
[SWS_VUCM_01303]	Dependencies between Software Clusters
[SWS_VUCM_01305]	Vehicle Package format
[SWS_VUCM_01306]	TransferExit Invalid package manifest
[SWS_VUCM_01307]	Vehicle Package format not supported
[SWS_VUCM_01308]	Check Vehicle Package version compatibility against V-UCM version
[SWS_VUCM_01309]	Definition of ImplementationDataType VehicleConditionCollectionType
[SWS_VUCM_01310]	Definition of ImplementationDataType VehicleConditionType
[SWS_VUCM_01311]	Semantic versioning

**Table G.1: Added Specification Items in R23-11**

### G.1.2 Changed Specification Items in R23-11

none

### G.1.3 Deleted Specification Items in R23-11

none

#### G.1.4 Added Constraints in R23-11

Number	Heading
[SWS_VUCM_CONSTR_00003]	Exclusive use of Vehicle Driver Interface
[SWS_VUCM_CONSTR_00004]	Unsupported safety by Vehicle driver interface
[SWS_VUCM_CONSTR_00005]	Safety state change
[SWS_VUCM_CONSTR_00006]	Exclusive use of Vehicle State Manager
[SWS_VUCM_CONSTR_00007]	Unsupported safety conditions by Vehicle State Manager
[SWS_VUCM_CONSTR_00009]	Safety condition change
[SWS_VUCM_CONSTR_00011]	Flashing Adapter provided interface
[SWS_VUCM_CONSTR_00013]	Confidential information protection
[SWS_VUCM_CONSTR_00015]	Trigger on <code>kVehicleChecking</code> state
[SWS_VUCM_CONSTR_00016]	OTA Client use of RequestedPackage field
[SWS_VUCM_CONSTR_00017]	Interaction of <code>V-UCM</code> with Vehicle Driver
[SWS_VUCM_CONSTR_00018]	V-UCM uniqueness in vehicle

**Table G.2: Added Constraints in R23-11**

#### G.1.5 Changed Constraints in R23-11

none

#### G.1.6 Deleted Constraints in R23-11

none