# Contents

1. [Introduction and functional overview](#_bookmark0) 9
2. Acronyms and abbreviations 10
3. [Related documentation](#_bookmark1) 12
   1. [Input documents & related standards and norms](#_bookmark2) 12
   2. [Related specification](#_bookmark14) 12
   3. [Further applicable specification](#_bookmark15) 13
4. [Constraints and assumptions](#_bookmark16) 14
   1. [Known Limitations](#_bookmark17) 14
   2. [Applicability to car domains](#_bookmark18) 14
5. [Dependencies to other functional clusters](#_bookmark19) 15
   1. [Interfaces to Adaptive State Management](#_bookmark20) 15
   2. [UCM service over ara::com](#_bookmark21) 15
   3. [Interfaces to Adaptive Crypto Interface](#_bookmark22) 15
   4. [Interfaces to Identity and Access Management](#_bookmark23) 16
   5. [Interfaces to Adaptive TimeSynchronization Interface](#_bookmark24) 16
   6. [UCM use of Persistency library](#_bookmark25) 16
6. Requirements Tracing 17
7. [Functional specification](#_bookmark26) 28
   1. [UCM](#_bookmark27) 28
      1. [Software Cluster lifecycle](#_bookmark28) 28
      2. [Technical Overview](#_bookmark30) 29
         1. [Software Package Management](#_bookmark32) 30
         2. [Runtime dependencies](#_bookmark37) 34
         3. [Update scope and State Management](#_bookmark38) 34
      3. [Transferring Software Packages](#_bookmark40) 35
         1. [Error handling in TransferStart](#_bookmark43) 38
         2. [Error handling in TransferData](#_bookmark44) 38
         3. [Error handling in TransferExit](#_bookmark55) 40
         4. [Error handling in DeleteTransfer](#_bookmark61) 42
      4. [Processing of Software Packages from a stream](#_bookmark64) 42
      5. [Processing Software Packages](#_bookmark65) 43
         1. [Error handling during Processing Software Packages](#_bookmark66) 44
         2. [Error handling for Cancel](#_bookmark78) 46
         3. [Error handling for RevertProcessedSwPackages](#_bookmark81) 47
         4. [Error handling for GetSwProcessProgress](#_bookmark84) 47
      6. [Activation and Rollback](#_bookmark85) 47
         1. [Activation](#_bookmark86) 48
         2. [Rollback](#_bookmark91) 50
         3. [Boot options](#_bookmark93) 51
         4. [Finishing activation](#_bookmark94) 52
      7. [Status Reporting](#_bookmark95) 52
      8. [Robustness against reset](#_bookmark97) 57
         1. [Boot monitoring](#_bookmark98) 57
      9. [History](#_bookmark99) 58
      10. [Version Reporting](#_bookmark100) 58
      11. [Securing Software Updates](#_bookmark101) 59
      12. [Functional cluster lifecycle](#_bookmark103) 59
          1. [Shutdown behaviour](#_bookmark104) 60
   2. [UCM Master](#_bookmark105) 61
      1. [UCM Master Functional Cluster lifecycle](#_bookmark106) 61
      2. [Technical Overview](#_bookmark108) 61
      3. [UCM Master general behaviour](#_bookmark109) 62
      4. [UCM identification](#_bookmark110) 63
      5. [UCM Master Software Packages transfer or streaming](#_bookmark111) 63
      6. [Adaptive Applications interacting with UCM Master](#_bookmark112) 64
         1. [OTA Client](#_bookmark113) 65
         2. [Vehicle Driver Interface](#_bookmark114) 66
         3. [Vehicle State Manager](#_bookmark115) 68
         4. [Flashing Adapter](#_bookmark116) 69
      7. [Non Adaptive Platform update](#_bookmark117) 70
         1. [D-PDU API implementation support](#_bookmark118) 70
         2. [Not required D-PDU API concepts](#_bookmark119) 71
         3. [Not required D-PDU API functions](#_bookmark120) 71
         4. [Classic platform update with UCM Master and diag-](#_bookmark121) [nostic tool](#_bookmark121) 73
      8. [Status reporting](#_bookmark122) 74
         1. [States](#_bookmark125) 76
         2. [States Transitions](#_bookmark126) 77
      9. [Campaign cancelling](#_bookmark127) 79
      10. [Campaign Reporting](#_bookmark128) 80
      11. [Content of Vehicle Package](#_bookmark129) 81
      12. [Vehicle update security and confidentiality](#_bookmark130) 83
8. [API specification](#_bookmark131) 84
9. [Service Interfaces](#_bookmark132) 85
   1. [Type definitions](#_bookmark133) 85
      1. UCMIdentifierType 87
      2. TransferIdType 87
      3. SwNameType 87
      4. StrongRevisionLabelString 88
      5. SwNameVersionType 88
      6. SwNameVersionVectorType 88
      7. ByteVectorType 89
      8. SwPackageStateType 89
      9. SwPackageInfoType 89
      10. SwPackageInfoVectorType 90
      11. SwDescType 90
      12. SwDescVectorType 91
      13. SwPackageDescType 91
      14. SwPackageDescVectorType 91
      15. SwClusterStateType 92
      16. SwClusterInfoType 92
      17. SwClusterInfoVectorType 92
      18. PackageManagementStatusType 93
      19. ActionType 93
      20. ResultType 94
      21. GetHistoryType 94
      22. GetHistoryVectorType 94
      23. CampaignHistoryType 95
      24. CampaignHistoryVectorType 95
      25. CampaignResultType 95
      26. UCMMasterResolutionType 96
      27. UCMStepErrorVectorType 96
      28. UCMStepErrorType 96
      29. SoftwarePackageStepType 97
      30. HistoryVectorType 97
      31. HistoryType 98
      32. CampaignStateType 98
      33. TransferStateType 99
      34. SafetyConditionType 99
      35. SafetyConditionsVectorType 99
      36. SafetyStatesType 100
      37. SafetyStatesVectorType 100
   2. Provided Service Interfaces 100
      1. Package Management 100
      2. Vehicle Package Management 107
      3. Vehicle Driver Application Interface 115
      4. Vehicle State Manager 118
   3. Required Interface 120
      1. State Management Update Request 120
   4. Application Errors 120
      1. Application Error Domain 120
         1. UCMErrorDomain 120
10. Sequence diagrams 122
    1. Update process 122
    2. Data transmission 123
    3. Package processing 124
    4. Activation 125
    5. Failing activation 126
    6. Failing rollback 127
    7. UCM Master simplified vehicle update 128
11. Mentioned Manifest Elements 129
12. [Interfaces to other Functional Clusters (informative)](#_bookmark134) 138
    1. [Overview](#_bookmark135) 138
    2. [Interfaces Tables](#_bookmark136) 138
       1. [UCM update notification](#_bookmark137) 138
13. [Packages distribution within vehicle detailed sequence examples](#_bookmark138) 139
    1. [Collect information of present Software Clusters in vehicle](#_bookmark139) 139
    2. [Action computation](#_bookmark140) 139
       1. [Pull package from Backend into vehicle](#_bookmark141) 140
       2. [Push package from backend into vehicle](#_bookmark142) 140
    3. [Packages transfer from backend into targeted UCM](#_bookmark143) 142
    4. [Package processing](#_bookmark144) 143
    5. [Package activation](#_bookmark145) 144
    6. [Package rollback](#_bookmark146) 145
    7. [Campaign reporting](#_bookmark147) 146
14. [Security Analysis of Installation and Update](#_bookmark148) 147
    1. [Securing Software Package](#_bookmark149) 147
    2. [Securing Calls to UCM](#_bookmark150) 147
    3. [Suppressing Call to UCM](#_bookmark151) 148
    4. [Resource Starvation](#_bookmark152) 148
    5. [Zombie Sessions](#_bookmark153) 148
15. History of Constraints and Specification Items 150
    1. Constraint and Specification Item History of this document according

to AUTOSAR Release R19-11. 150

* + 1. Added Traceables in R19-11 150
    2. Changed Traceables in R19-11 153
    3. Deleted Traceables in R19-11 153
    4. Added Constraints in R19-11 154
    5. Changed Constraints in R19-11 154
    6. Deleted Constraints in R19-11 154
  1. Constraint and Specification Item History of this document according

to AUTOSAR Release R20-11. 154

* + 1. Added Traceables in R20-11 154
    2. Changed Traceables in R20-11 157
    3. Deleted Traceables in R20-11 160
    4. Added Constraints in R20-11 161
    5. Changed Constraints in R20-11 161
    6. Deleted Constraints in R20-11 162
  1. Constraint and Specification Item History of this document according

to AUTOSAR Release R21-11. 162

* + 1. Added Traceables in R21-11 162
    2. Changed Traceables in R21-11 163
    3. Deleted Traceables in R21-11 167
    4. Added Constraints in R21-11 168
    5. Changed Constraints in R21-11 168
    6. Deleted Constraints in R21-11 168
  1. Constraint and Specification Item History of this document according

to AUTOSAR Release R22-11. 168

* + 1. Added Traceables in R22-11 168
    2. Changed Traceables in R22-11 169
    3. Deleted Traceables in R22-11 173
    4. Added Constraints in R22-11 173
    5. Changed Constraints in R22-11 174
    6. Deleted Constraints in R22-11 174

# 1 Introduction and functional overview

This software specification contains the functional description and interfaces of the functional cluster Update and Configuration Management which belongs to the AUTOSAR Adaptive Platform Services. Update and Configuration Man- agement has the responsibility of installing, updating and removing software on an AUTOSAR Adaptive Platform in a safe and secure way while not sacrificing the dynamic nature of the AUTOSAR Adaptive Platform.

The Update and Configuration Management functional cluster is responsible for:

* Version reporting of the software present in the AUTOSAR Adaptive Platform
* Receiving and buffering software updates
* Checking that enough resources are available to ensure a software update
* Performing software updates and providing log messages and progress informa- tion
* Validating the outcome of a software update
* Providing rollback functionality to restore a known functional state in case of fail- ure

In addition to updating and changing software on the AUTOSAR Adaptive Plat- form, the Update and Configuration Management is also responsible for up- dates and changes to the AUTOSAR Adaptive Platform itself, including all func- tional clusters, the underlying POSIX OS and its kernel with the responsibilities defined above.

In order to allow flexibility in how Update and Configuration Management is used, it will expose its functionality via ara::com service interfaces, not direct APIs. This ensures that the user of the functional cluster Update and Configuration Management does not have to be located on the same ECU.

# Related documentation

## Input documents & related standards and norms

1. Glossary AUTOSAR\_TR\_Glossary
2. Specification of Execution Management AUTOSAR\_SWS\_ExecutionManagement
3. Specification of State Management AUTOSAR\_SWS\_StateManagement
4. Specification of Communication Management AUTOSAR\_SWS\_CommunicationManagement
5. General Requirements specific to Adaptive Platform AUTOSAR\_RS\_General
6. Specification of Cryptography AUTOSAR\_SWS\_Cryptography
7. Specification of Identity and Access Management AUTOSAR\_SWS\_IdentityAndAccessManagement
8. Specification of Time Synchronization AUTOSAR\_SWS\_TimeSynchronization
9. Requirements on Update and Configuration Management AUTOSAR\_RS\_UpdateAndConfigurationManagement
10. Specification of Manifest AUTOSAR\_TPS\_ManifestSpecification
11. Explanation of Adaptive Platform Design AUTOSAR\_EXP\_PlatformDesign
12. Specification of Persistency AUTOSAR\_SWS\_Persistency
13. Specification of Platform Health Management AUTOSAR\_SWS\_PlatformHealthManagement

## Related specification

See chapter [3.1](#_bookmark2).

## Further applicable specification

AUTOSAR provides a general specification [[5](#_bookmark6)] which is also applicable for UCM. The specification RS General shall be considered as additional and required specification for implementation of UCM.

# Constraints and assumptions

## Known Limitations

UCM is not responsible to initiate the update process. UCM realizes a service interface to achieve this operation. The user of this service interface is responsible to verify that the vehicle is in a updatable state before executing a software update procedure on demand. It is also in the responsibility of the user to communicate with other AUTOSAR Adaptive Platforms or AUTOSAR Classic Platforms within the vehicle.

The UCM receives a locally available software package for processing. The software package is usually downloaded from the OEM backend. The download of the software packages has to be done by another application, i.e. UCM does not manage the connec- tion to the OEM backend. Prior to triggering their processing, the software packages have to be transferred to UCM by using the provided ara::com interface.

The UCM update process is designed to cover updates on use case with single AUTOSAR Adaptive Platform. UCM can update Adaptive Applications, the AUTOSAR Adaptive Platform itself, including all functional clusters and the under- lying OS.

The UCM is not responsible for enforcing authentication and access control to the pro- vided interfaces. The document currently does not provide any mechanism for the confidentiality protection as well as measures against denial of service attacks. The assumption is that the platform preserves the integrity of parameters exchanged be- tween UCM and its user.

The possibility to restart a specific application instead of a Machine reboot depends of the kind of update and application, is therefore implementation specific and is defined in the Software Package manifest.

UCM does only support updates of ARA::COM and UDS (ISO-14229) compliant ECUs. UCM is not controlling any action done by diagnostic tool directly updating a Classic plat- form. For instance UCM cannot protect against downgrading of a Software Cluster in a Classic platform by a diagnostic tool.

## Applicability to car domains

No restrictions to applicability.

# Dependencies to other functional clusters

The UCM functional cluster expose services to client applications via the ara::com

middleware.

SoftwareCluster



Software Package A

Signed container

UCM Client

ara::com

Software Package Manifest

Authentication tag

Dependencies to Functional Clusters

Identity & Access Management

Posix

State Management

Crypto API

Persistency

|  |
| --- |
| Executables |
| Data |
| Manifests |

**Figure 5.1: UCM dependencies to other Functional Clusters.**

## Interfaces to Adaptive State Management

UCM relies on State Management and its provided UpdateRequest Service Inter- face to perform the necessary Function Group state changes needed to activate the newly installed, updated or removed software.

Certain applications can conflict with the update process or the newly updated pack- age, and they need to be stopped during the update process. This could be achieved by putting the machine to a safe Machine state, by activating a combination of suit- able Function Groups and its states. It is the responsibility of the platform integrator to define this state or Function Groups. The Adaptive Application accessing the UCM, should make sure that the platform is switched to this state (using interfaces from State Management), before starting the update.

## UCM service over ara::com

The UCM shall provide a service interface over ara::com using methods and fields.

## Interfaces to Adaptive Crypto Interface

UCM uses Crypto Interface for AUTOSAR Adaptive Platform [[6](#_bookmark7)] to verify package integrity and authenticity and to decrypt confidential update data.

## Interfaces to Identity and Access Management

Identity and Access Management [[7](#_bookmark8)] controls the UCM’s Clients access to UCM’s service interface PackageManagement.

## Interfaces to Adaptive TimeSynchronization Interface

UCM uses TSync Interface [[8](#_bookmark9)] for AUTOSAR Adaptive Platform to get the latest timestamp.

## UCM use of Persistency library

UCM may use ara::per to store internal status information. This can e.g. be used to recover after reboot.

# Functional specification

## UCM

### Software Cluster lifecycle

RevertProcessedSw Packages



Final

Finish (from kRolledBack)

Initi al

**UPDATING**

Finish (from kActivated)

ProcessSw Package

ProcessSw PackageFinish (from

k Activated)

RevertProcessedSw Packages

ProcessSw Package [actionType == remove]

**ADDED**

Finish (from

k Activated)

**PRESENT**

**REMOVED**

Finish (from kRolledBack)

RevertProcessedSw Packages



**Figure 7.1:** **State Machine for a Software Cluster**

The state machine in Fig. [7.1](#_bookmark29) describes the life-cycle states of a Software Cluster. These states are reported with GetSwClusterChangeInfo method.

**[SWS\_UCM\_00191] Software Cluster life-cycle state kAdded** *[*A Software Cluster state shall be kAdded after the Software Cluster is successfully pro- cessed with ProcessSwPackage method call on the AUTOSAR Adaptive Plat- form and if it was not previously present in the AUTOSAR Adaptive Platform and before activation is finished.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00192] Software Cluster life-cycle state transition from kAdded to kPresent** *[*A Software Cluster state shall change from kAdded to kPre- sent after a successful activation of a newly added Software Cluster with Finish method call.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00195] Software Cluster life-cycle state kUpdating** *[*A Software Cluster state shall be kUpdating after a successful processing of the updated Software Cluster with ProcessSwPackage method call and before activation is finished.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00193] Software Cluster life-cycle state transition from kUpdat- ing to kPresent** *[*A Software Cluster state shall change from kUpdating to kPresent after a successful activation of the updated Software Cluster with Finish method call, or after reverting the Software Cluster update with a Re- vertProcessedSwPackages method call.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00196] Software Cluster life-cycle state kRemoved** *[*A Software Cluster state shall be kRemoved after successful completion of method ProcessS- wPackage which involves the removal of the existed Software Cluster and before activation is finished.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00194] Software Cluster life-cycle state transition from kRe- moved to kPresent in case of RevertProcessedSwPackages call** *[*A Software Cluster state shall change from kRemoved to kPresent after a successful call to RevertProcessedSwPackages method in case the Software Cluster was pre- viously requested to be removed by ProcessSwPackage method call.*♩(RS\_UCM\_- 00011)*

**[SWS\_UCM\_00286] Software Cluster life-cycle state transition from kRe- moved to kPresent in case of Finish call** *[*A Software Cluster state shall change from kRemoved to kPresent after a successful call to Finish method in case a Software Cluster being removed has to be rolled back after a failing acti- vation.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00197] End of Software Cluster life-cycle state from state kAdded in case of RevertProcessedSwPackages call** *[*A Software Cluster shall reach the end of its life-cycle from kAdded after a successful removal of a newly added Software Cluster with RevertProcessedSwPackages method call in case the Software Cluster was previously requested to be added by ProcessSwPackage method call.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00287] End of Software Cluster life-cycle state from state kAdded in case of Finish call** *[*A Software Cluster shall reach the end of its life-cycle from kAdded after a successful removal of a newly added Software Cluster with Finish method call in case the newly added Software Cluster has to be rolled back after a failing activation.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00198] End of Software Cluster life-cycle state from state kRe- moved** *[*A Software Cluster shall reach the end of its life-cycle if it is successfully removed with a Finish method call and the Software Cluster is in state kRe- moved.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00199] Reporting of Software Cluster reaching end of life-cycle** *[*Any Software Cluster reaching the end of its life-cycle shall not be reported by UCM any more.*♩(RS\_UCM\_00011)*

### Technical Overview

One of the declared goals of AUTOSAR Adaptive Platform is the ability to flexibly update the software and its configuration through over-the-air updates. During the life- cycle of an AUTOSAR Adaptive Platform, UCM is responsible to perform software modifications on the machine and to retain consistency of the whole system.

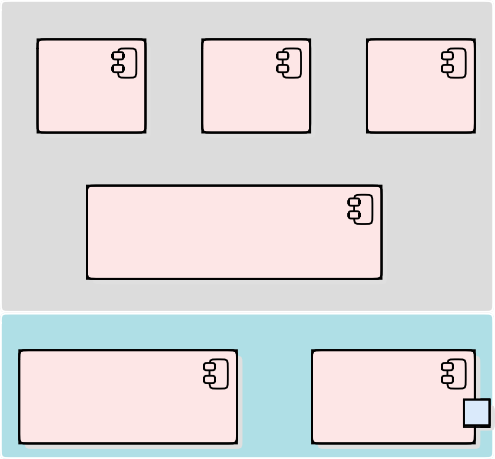
The UCM Functional Cluster provides a service interface that exposes its func- tionality to retrieve AUTOSAR Adaptive Platform software information and consis- tently execute software updates. Since ara::com is used, the client using the UCM service interface can be located on the same AUTOSAR Adaptive Platform, but also remote clients are possible.

The service interface has been primarily designed with the goal to make it possible to use standard diagnostic services for downloading and installing software updates for the AUTOSAR Adaptive Platform. However, the methods and fields in the service interface are designed in such a way that they can be used in principle by any Adaptive Application. UCM does not impose any specific protocol on how data is transferred to the AUTOSAR Adaptive Platform and how package processing is controlled. In particular UCM does not expose diagnostic services.

As shown in Figure [7.2](#_bookmark31), whether the use case is an over-the-air update or garage up- date done through diagnostics, it is not visible to the UCM. The UCM Client abstracts the use case from the UCM and forwards the data stream and sequence control commands to the UCM. Later in this document, the term UCM Client is used to describe an Adap- tive Application that consumes UCM PackageManagement services through UCM ara::com API. Diagnostic Application and UCM Master are two examples of such UCM Clients.

**Vehicle**

«device»



**Adaptive ECU**

**AUTOSAR Adaptive Application Layer**

**Server**

**App A App B App ...**

**UCM Client (Diagnostic**

**Application / UCM Master)**

«optional»

**Cloud**

**AUTOSAR Adaptive Platform Services + Foundation**

«ServiceProvi...

**UCM**

**Diagnostic Manager (DM)**

DoIP socket

«optional»

**Diagnostic Client**

**Figure 7.2:** **Architecture overview for diagnostic use case**

### Software Package Management

The UCM update sequence consists three different phases:

* Software Package transfer: A phase in which, one or several Software Packages are transferred from the UCM’s Client Application to the internal buffer of the UCM. For further information see chapter [7.1.3](#_bookmark40).
* Software Package processing: A phase in which the UCM performs the oper- ation (kInstall, kUpdate, kRemove) on the relevant SoftwareCluster. For further information see chapter [7.1.5](#_bookmark65).
* Activation: A phase in which the UCM checks the dependencies of the Soft- wareClusters that have been involved in the operation, then activates them and finally check that all the SoftwareClusters can be executed properly (via State Management) prior to finishing the update. For further information see chapter [7.1.6](#_bookmark85)

### Software Package

**[SWS\_UCM\_00122] Software Package utilization** *[*The unit for deployment that the UCM shall take as input is called Software Package, see [[1](#_bookmark3)]. Each Software Package shall address a single SoftwareCluster.*♩(RS\_UCM\_00026)*

A SoftwareCluster can act in two roles:

‘Sub’-SoftwareCluster : It is a SoftwareCluster without diagnostic target address, containing processes, executables and further elements

‘Root’-SoftwareCluster : It is a SoftwareCluster with a diagnostic target address that may reference several other ‘Sub’-SoftwareClusters, which thus form a logical group.

A SoftwareCluster can be of the following categories expressed by the attribute

SoftwareCluster.category :

APPLICATION\_LAYER: the SoftwareCluster can be removed by UCM

PLATFORM\_CORE: the SoftwareCluster cannot be removed as it would break the system.

PLATFORM: the SoftwareCluster is part of the platform software and can be removed

**[SWS\_UCM\_00245]**{DRAFT} **Software Cluster category** *[*UCM shall not remove a SoftwareCluster that has installationBehavior set to value cannotBeRe- moved. In case of such an attempt, UCM shall raise ApplicationError SwclRe- movalDenied.*♩(RS\_UCM\_00028, RS\_UCM\_00029)*

A Software Package has to be modelled as a so-called SoftwareCluster which describes the content of a Software Package that is downloaded or uploaded to the AUTOSAR Adaptive Platform, see [[10](#_bookmark10)].

The term Software Package is used for the "physical", uploadable Software Package that is processed by UCM whereas the term SoftwareCluster is used for the modeling element. In the model, the content of a SoftwareCluster is de- fine by references to all required model elements. The SoftwareCluster and the related model elements define the content of the manifest that is part of the Software Package. The Software Package format and the update scope are described in chapter "Content of a Software Package" as well as in [[11](#_bookmark11)].

**[SWS\_UCM\_CONSTR\_00012]**{DRAFT} *[*The SoftwareCluster aggregation of ArtifactChecksum shall not include the uri of this same SoftwareCluster mani- fest.*♩(RS\_UCM\_00012)*

The uri attribute in ArtifactChecksum is referring to the artifact contained in the

SoftwareCluster.

### Content of a Software Package

Each Software Package addresses a single SoftwareCluster and contains manifests, executables and further data (depending on the role of the SoftwareClus- ter) as the example sketched in Figure [7.3](#_bookmark35).

Software Package A Signed container

SoftwareCluster A

Signed container

Software Cluster Manifest

Authentication tag

Software Package Manifest

Authentication tag

Manifests

Data

Executables

**Figure 7.3:** **Software Package content description**

A single Software Package is designed in a way that it could contain one or several executables of Adaptive Applications, kernel or firmware updates, or updated configuration and calibration data to be deployed on the AUTOSAR Adaptive Plat- form.

The Software Package manifest is recommended to be sent at the beginning in order for UCM to have early information of for instance memory usage or streaming.

An exemplary implementation of the adaptive workflow with Software Packages can be seen in chapter Methodology and Manifest in [[11](#_bookmark11)]. For more details on the Software Package class, you can refer to SoftwarePackage

**[SWS\_UCM\_00112] Software Cluster and version** *[*SoftwareCluster’s mani- fest shall include a name and a version following description of StrongRevisionLa- belString.*♩(RS\_UCM\_00002)*

**[SWS\_UCM\_CONSTR\_00001]** *[*If any content (for instance an executable or persis- tent data) of an already installed SoftwareCluster is modified by an incoming Software Package, then the version number of the incoming SoftwareCluster indicated in the Software Package shall be higher than the version number of the al- ready installed SoftwareCluster.*♩(RS\_UCM\_00002, RS\_UCM\_00010, RS\_UCM\_- 00011)*

If the constraint is violated, an error will be raised according to [[SWS\_UCM\_00103](#_bookmark102)].

A higher version number is achieved by an increment of the MajorVersion, the Mi- norVersion, or the PatchVersion.

If there is a need to downgrade a failing SoftwareCluster (for instance, malfunction in the field that was not detected at activation), it will therefore be needed to repackage the same old SoftwareCluster that was properly working with an higher version number.

**[SWS\_UCM\_00130] Software Cluster and version error** *[*If SoftwareClus- ter’s manifest does not contain any SoftwareCluster.version following descrip- tion of StrongRevisionLabelString, UCM shall raise the ApplicationError InvalidPackageManifest.*♩(RS\_UCM\_00002)*

**[SWS\_UCM\_CONSTR\_00014]**{DRAFT} **Software Package and Software Clus- ter shortNames** *[*SoftwarePackage and the referenced SoftwareCluster shall share the same shortName in order to be able to compare their versions.*♩(RS\_UCM\_- 00033)*

### Applications Persisted Data

Updating and rolling back of persisted data is handled completely by the application using persistency without involvement of UCM. A detailed explanation can be found in the Persistency Specification [[12](#_bookmark12)]. An exception here is the removal of persistent data after a SoftwareCluster is removed.

**[SWS\_UCM\_00184] Persistent data clean-up after Software Cluster removal** *[*UCM shall remove persistent data of a removed SoftwareCluster by aggregating the information given in the application manifest, namely PersistencyKeyValueStorage.uri and PersistencyFileStorage.uri, in order to leave the AUTOSAR Adaptive Platform and the file system clean.*♩(RS\_UCM\_00026, RS\_UCM\_00005, RS\_UCM\_00004)*

**[SWS\_UCM\_00273]**{DRAFT} **Persistent data clean-up after Software Cluster up- date that removes a process** *[*UCM shall remove persistent data of a removed pro- cess by aggregating the information given in the execution manifest, namely Per- sistencyKeyValueStorage.uri and PersistencyFileStorage.uri, in order to leave the AUTOSAR Adaptive Platform and the file system clean.*♩(RS\_UCM\_00026, RS\_- UCM\_00005, RS\_UCM\_00004)*

Persistent data can include administrative and backup data.

### Runtime dependencies

Processes within a SoftwareCluster can have functional dependencies toward other SoftwareClusters.

Dependencies are described in the SoftwareCluster metamodel, see [[10](#_bookmark10)].

**[SWS\_UCM\_00120]**{DRAFT} **Runtime dependencies check** *[*Before the activation of the new software version triggered by Activate method call, UCM shall check runtime dependencies.*♩(RS\_UCM\_00007)*

The rationale is, if UCM has to process several Software Packages, then execu- tion dependencies may not be fulfilled at all times during the Software Packages process but must be fulfilled before changes can be activated.

### Update scope and State Management

Software Package processed by UCM can contain Adaptive Applications, up- dates to AUTOSAR Adaptive Platform itself or to the underlying OS. Update type depends on the content of the Software Package.

**[SWS\_UCM\_00099] Update of Adaptive Application** *[*UCM shall be able to up- date Adaptive Applications*♩(RS\_UCM\_00001)*

**[SWS\_UCM\_00100]**{DRAFT} **Update of Functional Clusters** *[*UCM shall be able to update all Functional Clusters, including UCM itself.*♩(RS\_UCM\_00028)*

**[SWS\_UCM\_00101]**{DRAFT} **Update of Host** *[*UCM shall be able to update the under- lying OS hosting the AUTOSAR Adaptive Platform.*♩(RS\_UCM\_00029)*

Definition of an updatable state with respect to the system setup is the OEM respon- sibility. Based on the system setup and the application, the system might need to be

switched into a predefined state, to free resource to speed up the update, to block nor- mal usage of software which might cause interruptions to update process and to block using functionality which might be interrupted by the update sequence.

**[SWS\_UCM\_00257] Update session** *[*To confirm the system is in an updatable state, UCM shall start an update session by calling State Management UpdateRequest Service Interface RequestUpdateSession method after its dependency check trig- gered by Activate method call successfully completes.*♩(RS\_UCM\_00026, RS\_- UCM\_00003)*

**[SWS\_UCM\_00258] Update session rejected** *[*If State Management Up- dateRequest Service Interface RequestUpdateSession method call raises er- ror kRejected, UCM shall transition from kActivating to kReady states and Ac- tivate method call shall return ApplicationError UpdateSessionRejected.*♩ (RS\_UCM\_00026, RS\_UCM\_00024)*

If update session could be recurrently rejected, it is up to implementer to cache the dependency check result in order to avoid unnecessary computation and compute it only once.

During the update session, the minimum applications required for the Update process should be executed. This way system is more robust, more resources are free and user is blocked from using applications, of which failure could cause safety risk to the user.

Update of some components require a Machine reset to be performed. These com- ponents should be configured to be part of Function Group MachineFG, as the update sequence of Function Group MachineFG includes a Machine reset. Ex- ecution Management, State Management, Communication Management and UCM itself are good examples which probably require a Machine reset to activate the up- date. Other such components could be applications involved in the update sequence or applications involved in safety monitoring. Further details on Function Group MachineFG can be found in State Management.

### Transferring Software Packages

To speed up the overall data transmission time, the package transfer is decoupled from the processing and activation process. This section describes requirements for initiation of a data transfer, the data transmission and ending of the data transmission.

Each Software Package gets its own state as soon as it is being transferred to UCM. The state machines in Fig. [7.4](#_bookmark41) specify the lifecycle of a Software Package that is transferred to and processed by UCM. During this lifecycle, a Software Package is uniquely identified with an id that UCM provides to the client.

The UCM has the possibility to keep the Software Package in kTransferred states in case it failed and retry later: transferring Software Package can be costly, if

it is authenticated, there could be no reason to delete it if the update has not been successfully finished.

Initi al



TransferStart

TransferData

**kTransferring kTransferred**

TransferExit

ProcessSw Package

**kProcessing**

Cancel

[TransferCompleted]

[TransferNotCompleted]

DeleteTransfer

[transfer blocks not stored]

RevertProcessedSw Packages

Transfer Completed ?

ProcessSwPackage

D eleteTransfer

[transfer blocks not stored]

[ProcessSwPackageDone]

[transfer blocks stored]

[transfer blocks not stored]

Final

Stored transfer blocks ?

Stored transfer blocks ?

D eleteTransfer

Finish, RevertProcessedSw Packages

RevertProcessedSwPackages, ProcessedSwPackageCancelled

**kProcessingStream kProcessed**

[ProcessSw PackageDone]

TransferData, TransferExit

**Figure 7.4:** **State Machine representing Software Packages lifecycle**

**[SWS\_UCM\_00007] Data transfer at any time** *[*UCM shall provide support to trans- fer Software Packages at any time when UCM is running. Transferring is decou- pled from the UCM Package Management states.*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

**[SWS\_UCM\_00272]**{DRAFT} **Transfer block size** *[*TransferStart shall return BlockSize parameter to indicate the maximum block size (unit: bytes, as defined by maxBlockSize) to be allowed to transfer in one TransferData method call.*♩ (RS\_UCM\_00025)*

The block size should be aligned to flashing capability in case of Classic Platform capability for instance.

**[SWS\_UCM\_00088] Preparation of data transfer** *[*Data transfer shall be prepared with the method TransferStart. In the preparation step the number of bytes to be transferred is provided by the client and UCM assigns an id for the Software Package to be transferred.*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

While a Software Package is being transferred, if UCM receives a subsequent TransferStart call targeting another Software Package, UCM should make sure that the sum of the size of both Software Packages (the one being transferred and the one requested to be transferred) does not exceed the size of the UCM buffer. Oth- erwise, the TransferStart should raise the ApplicationError Insufficient- Memory and the newly requested transmission should be rejected as described above.

**[SWS\_UCM\_00008] Executing the data transfer** *[*After successful call of Trans- ferStart method, the transmission of the Software Package block-wise shall be supported by the method TransferData.*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_- UCM\_00025)*

**[SWS\_UCM\_00145] Sequential order of data transfer** *[*The method Transfer- Data shall support the parameter blockCounter that shall start with 0x01 and be incremented by one for each subsequent block.*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

**[SWS\_UCM\_00010] End of data transfer** *[*After transmission of a Software Pack- age is completed, the transmission can be finished with method TransferExit.*♩ (RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

Software Package contains authentication and integrity tags, which are used during the transfer sequence to authenticate the content of the Software Package.

**[SWS\_UCM\_00075] Multiple data transfers in parallel** *[*Handling of multiple data transfers in parallel shall be supported by UCM.*♩(RS\_UCM\_00019)*

If UCM provide enough buffering resources for Software Packages, several pack- ages could be transferred (in parallel) before they are processed one after the other. The processing (i.e. unpacking and actually applying changes to the AUTOSAR Adap- tive Platform) of Software Packages described by the state kProcessing is further detailed in Sect. [7.1.5](#_bookmark65).

**[SWS\_UCM\_00021] Deleting transferred Software Packages** *[*UCM shall provide a method DeleteTransfer that shall delete the targeted Software Package and free the resources reserved to store that Software Package.*♩(RS\_UCM\_00018)*

**[SWS\_UCM\_00069]**{OBSOLETE} **Report information on Software Packages** *[* UCM shall provide a method GetSwPackages of the interface service PackageMan- agement to provide the Software Packages’ identifiers, names, versions, states, consecutive bytes received and consecutive blocks received.*♩(RS\_UCM\_00010, RS\_- UCM\_00002)*

At the invocation of method GetSwPackages of the service interface PackageMan- agement, UCM returns the Software Packages’ identifiers, names, versions, states, consecutive bytes received and consecutive blocks received.

If Software Package is in kTransferring state, it is not possible to get versions or names as manifest could not be complete or accessible, therefore method GetSw- Packages should return empty values except for TransferID, ConsecutiveBytesRe- ceived and ConsecutiveBlocksReceived at this particular state.

**[SWS\_UCM\_00216] Validity of TransferId** *[*The TransferId of a Software Package shall be invalidated for further use when it reaches final lifecycle state.*♩(RS\_UCM\_- 00019)*

### Error handling in TransferStart

TransferStart allocates resources for the client transfer.

**[SWS\_UCM\_00140] UCM insufficient memory** *[*TransferStart method shall raise the ApplicationError InsufficientMemory if the UCM buffer has not enough resources to store the corresponding Software Package.*♩(RS\_UCM\_00013, RS\_- UCM\_00019, RS\_UCM\_00025)*

### Error handling in TransferData

TransferData executes the following checks. It is recommended to follow the speci- fied order.

**[SWS\_UCM\_00275]**{DRAFT} **TransferData error handling order** *[*Transfer- Data method shall check the following error conditions and return the respective error code.

1. [[SWS\_UCM\_00208](#_bookmark45)]
2. [[SWS\_UCM\_00203](#_bookmark46)]
3. [[SWS\_UCM\_00204](#_bookmark47)]
4. [[SWS\_UCM\_00243](#_bookmark48)]
5. [[SWS\_UCM\_00205](#_bookmark49)]
6. [[SWS\_UCM\_00206](#_bookmark50)]
7. [[SWS\_UCM\_00289](#_bookmark51)]
8. [[SWS\_UCM\_00207](#_bookmark52)]
9. [[SWS\_UCM\_00294](#_bookmark53)]
10. [[SWS\_UCM\_00098](#_bookmark54)]
11. [[SWS\_UCM\_00092](#_bookmark59)]
12. [[SWS\_UCM\_00245](#_bookmark33)]
13. [[SWS\_UCM\_00103](#_bookmark102)]

*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

**[SWS\_UCM\_00208] TransferData OperationNotPermitted** *[*Calling Transfer- Data after calling TransferExit for a specific TransferId shall raise the error Ap- plicationError OperationNotPermitted*♩(RS\_UCM\_00019)*

**[SWS\_UCM\_00203] TransferData InvalidTransferId** *[*TransferData shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId (An ID that was not initiated by TransferStart or marked invalid by DeleteTrans- fer or RevertProcessedSwPackages) is sent by the client.*♩(RS\_UCM\_00019)*

**[SWS\_UCM\_00204] TransferData IncorrectBlock** *[*TransferData shall raise ApplicationError IncorrectBlock upon receipt of a block counter value that is successfully transmitted to UCM before or upon receipt of an unexpected block counter value.*♩(RS\_UCM\_00014, RS\_UCM\_00019)*

**[SWS\_UCM\_00243] Too big block size received by UCM** *[*In the case the received block size with TransferData exceeds the block size returned by TransferStart for the same TransferId, UCM shall raise the ApplicationError IncorrectBlock- Size.*♩(RS\_UCM\_00013, RS\_UCM\_00014, RS\_UCM\_00025)*

**[SWS\_UCM\_00205] TransferData IncorrectSize** *[*In case the transferred Software package size exceeds the provided size in TransferStart, TransferData shall raise ApplicationError IncorrectSize*♩(RS\_UCM\_00014, RS\_UCM\_00019)*

**[SWS\_UCM\_00206] TransferData InsufficientMemory** *[*TransferData shall raise the error ApplicationError InsufficientMemory if resources to store the Software Package ceased to exist during the transfer operation.*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

**[SWS\_UCM\_00289]**{DRAFT} **TransferData TransferFailed** *[*TransferData shall raise the error ApplicationError TransferFailed if UCM cannot persist trans- ferred block.*♩(RS\_UCM\_00013)*

**[SWS\_UCM\_00207]**{DRAFT} **TransferData BlockInconsistent** *[*If UCM checks con- sistency of Block for each TransferData, UCM shall raise the error Application- Error BlockInconsistent in case Consistency check for transferred block fails.*♩ (RS\_UCM\_00012)*

The BlockInconsistent error is intended to be used by the Flashing Adapter. The Flashing Adapter can calculate additional consistency information for each block inter- nally, e.g. a CRC32 checksum. It can then use UDS protocol to send block data and checksum to the target ECU. In case checksum verification fails, the Flashing Adapter can report the BlockInconsistent error to the UCM Master or diagnostic client application.

As described in section [7.1.2.1.2](#_bookmark34) and [[10](#_bookmark10)], each Software Package has an authen- tication tag CryptoServiceCertificate which protects integrity and authenticity.

Therefore additional consistency check information is not needed. If authentication check fails, AuthenticationFailed error is intended to be used instead.

**[SWS\_UCM\_00294]**{DRAFT} **Unsupported package format for UCM** *[*In the case the Software Package archiving format is not supported, UCM TransferData method shall return ApplicationError UnsupportedPackageFormat.*♩(RS\_- UCM\_00025)*

**[SWS\_UCM\_00098]**{DRAFT} **Software Package Authentication failure** *[*UCM shall raise the ApplicationError AuthenticationFailed, if the Software Package authentication check fails.*♩(RS\_UCM\_00006, RS\_UCM\_00019, RS\_UCM\_- 00025)*

This error can happen when TransferData, TransferExit and ProcessSwPack- age methods are called. When AuthenticationFailed error is raised, it is up to client to decide if a DeleteTransfer will be called or not. The behaviour may vary depending on the life cycle, meaning R&D phase or on the field phase.

TransferData checks the package version format in accordance to [[SWS\_UCM\_00161](#_bookmark70)] (IncompatiblePackageVersion).

TransferData checks if the Software Cluster to be removed has attribute in- stallationBehavior set to cannotBeRemoved. If this is the case, UCM shall not remove it in accordance to [[SWS\_UCM\_00245](#_bookmark33)].

TransferData checks if the Software Cluster version being updated is older than currently present in Machine in accordance to [[SWS\_UCM\_00103](#_bookmark102)] (OldVer- sion).

### Error handling in TransferExit

**[SWS\_UCM\_00276]**{DRAFT} **TransferExit error handling order** *[*Transfer- Exit method shall check the following error conditions and return the respective error code.

1. [[SWS\_UCM\_00148](#_bookmark56)]
2. [[SWS\_UCM\_00212](#_bookmark57)]
3. [[SWS\_UCM\_00087](#_bookmark58)]
4. [[SWS\_UCM\_00294](#_bookmark53)]
5. [[SWS\_UCM\_00098](#_bookmark54)]
6. [[SWS\_UCM\_00092](#_bookmark59)]
7. [[SWS\_UCM\_00161](#_bookmark70)]
8. [[SWS\_UCM\_00213](#_bookmark60)]
9. [[SWS\_UCM\_00245](#_bookmark33)]
10. [[SWS\_UCM\_00103](#_bookmark102)]

*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

**[SWS\_UCM\_00148] Transfer sequence order** *[*Calling TransferExit without call- ing TransferData at least once or after TransferExit is called for a specific Trans- ferID, shall raise the ApplicationError OperationNotPermitted.*♩(RS\_UCM\_- 00019)*

**[SWS\_UCM\_00212] TransferExit InvalidTransferId** *[*TransferExit shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId is sent by the client.*♩(RS\_UCM\_00019)*

**[SWS\_UCM\_00087] Insufficient amount of data transferred** *[*When Transfer- Exit method is called, UCM shall check if all blocks of the Software Package have been transferred according to the size parameter of TransferStart. If not UCM shall return ApplicationError InsufficientData.*♩(RS\_UCM\_00013, RS\_- UCM\_00019, RS\_UCM\_00025)*

TransferExit checks if the Software Package archiving format is supported in accordance to [[SWS\_UCM\_00294](#_bookmark53)] (UnsupportedPackageFormat).

TransferExit checks authentication in accordance to [[SWS\_UCM\_00098](#_bookmark54)] (Au- thenticationFailed).

**[SWS\_UCM\_00092] Software Package integrity** *[*When TransferData or Trans- ferExit method is called, UCM shall raise the ApplicationError PackageIncon- sistent if the Software Package integrity check fails. This Software Pack- age integrity check may be realized by the UCM via a Software Package Checksum check or via other mechanisms.*♩(RS\_UCM\_00012, RS\_UCM\_00006)*

TransferExit checks the package version format in accordance to [[SWS\_UCM\_00161](#_bookmark70)] (IncompatiblePackageVersion).

**[SWS\_UCM\_00213] TransferExit InvalidPackageManifest** *[*TransferExit shall raise the error ApplicationError InvalidPackageManifest upon receival of an invalid manifest.*♩(RS\_UCM\_00012)*

TransferExit checks if the Software Cluster to be removed is from cate- gory PLATFORM\_CORE. If this is the case, UCM shall not remove it in accordance to [[SWS\_UCM\_00245](#_bookmark33)].

TransferExit checks if the Software Cluster version being updated is older than currently present in Machine in accordance to [[SWS\_UCM\_00103](#_bookmark102)] (OldVer- sion).

### Error handling in DeleteTransfer

**[SWS\_UCM\_00283]**{DRAFT} **DeleteTransfer error handling order** *[*Delete- Transfer method shall check the following error conditions and return the respective error code.

1. [[SWS\_UCM\_00214](#_bookmark62)]
2. [[SWS\_UCM\_00215](#_bookmark63)]

*♩(RS\_UCM\_00013, RS\_UCM\_00019, RS\_UCM\_00025)*

DeleteTransfer checks if the supplied parameter TransferId is valid.

**[SWS\_UCM\_00214] DeleteTransfer InvalidTransferId** *[*DeleteTransfer shall raise the error ApplicationError InvalidTransferId in case an invalid Trans- ferId is sent by the client.*♩(RS\_UCM\_00019)*

**[SWS\_UCM\_00215] DeleteTransfer OperationNotPermitted** *[*Calling Delete- Transfer during processing or during the processing stream shall raise the error ApplicationError OperationNotPermitted.*♩(RS\_UCM\_00019)*

### Processing of Software Packages from a stream

It is also possible to process a Software Package while the transfer is still ongoing. The following requirements apply for this use case.

**[SWS\_UCM\_00165] Processing from stream** *[*The UCM may support calling ProcessSwPackage directly from stream without waiting to receive the Software Package completely.*♩(RS\_UCM\_00001, RS\_UCM\_00003, RS\_UCM\_00004, RS\_- UCM\_00025)*

**[SWS\_UCM\_00166] Processing from stream state** *[*If UCM supports processing from stream and is in state kIdle or kReady, the method ProcessSwPackage for a Software Package in state kTransferring shall set this Software Package to state kProcessingStream.*♩(RS\_UCM\_00024, RS\_UCM\_00025)*

**[SWS\_UCM\_00167]**{DRAFT} **Cancelling streamed packages** *[*When Cancel is called, UCM shall remove all temporary and processed data of a Software Pack- age in state kProcessingStream.*♩(RS\_UCM\_00020, RS\_UCM\_00025)*

**[SWS\_UCM\_00168] Transferring while processing from stream** *[*Software Package state shall remain in kProcessingStream when TransferData is called.*♩(RS\_UCM\_00024, RS\_UCM\_00025)*

**[SWS\_UCM\_00169] Finishing transfer while processing from stream** *[*Software Package state shall be set to kProcessed when TransferExit is called and the Software Package is completely processed.*♩(RS\_UCM\_00024, RS\_UCM\_00025)*

**[SWS\_UCM\_00200] Failing authentication** *[*UCM shall delete the Software Pack- age and its related data processed by ProcessSwPackage call if authentication is failing at TransferExit or ProcessSwPackage call.*♩(RS\_UCM\_00039, RS\_UCM\_- 00006)*

### Processing Software Packages

In contrast to package transmission, only one Software Package can be processed at the same time to ensure consistency of the system. In the following, a software or package processing can involve any combination of an installation, update or re- moval of applications, configuration data, calibration data or manifests. It is up to the vendor-specific metadata inside a Software Package to describe the tasks UCM has to perform for its processing. For a removal, this might involve metadata describing which data needs to be deleted. Nevertheless, the communication sequence between the triggering application of the software modification and UCM is the same in any case. For an update of an existing application, the Software Package can contain only partial data, e.g. just an updated version of the execution manifest. Any UCM Client need to confirm that UCM is in kIdle CurrentStatus state before starting any up- date (process/activate).

**[SWS\_UCM\_00001]**{OBSOLETE} **Starting the package processing** *[*UCM shall pro- vide a method ProcessSwPackage to process transferred Software Package. id corresponding to Software Package shall be provided for this method.*♩(RS\_UCM\_- 00001, RS\_UCM\_00004)*

At the invocation of method ProcessSwPackage, UCM processes transferred Soft- ware Package with id argument corresponding to this Software Package.

**[SWS\_UCM\_00137] Processing several update Software Packages** *[*UCM shall support processing of several Software Packages, not in parallel, by calling method ProcessSwPackage several times in sequence.*♩(RS\_UCM\_00001, RS\_- UCM\_00004)*

During package processing, the progress is provided.

**[SWS\_UCM\_00018]**{OBSOLETE} **Providing Progress Information** *[*UCM shall pro- vide a method GetSwProcessProgress to query the progress of executing the Pro- cessSwPackage method call for provided TransferId. Parameter progress shall be set to a value representing the progress between 0% and 100% (0x00 ... 0x64).*♩(RS\_- UCM\_00023)*

At the invocation of method GetSwProcessProgress, UCM returns the progress of executing the ProcessSwPackage method call for provided TransferId. Parameter progress will be set to a value representing the progress between 0% and 100% (0x00 ... 0x64).

**[SWS\_UCM\_00003] Cancelling the package processing** *[*On call of Cancel

method, UCM shall abort the running package processing task, undo the changes to

the Software Cluster for which processing started and free the reserved resources used for it.*♩(RS\_UCM\_00020)*

**[SWS\_UCM\_00024] Revert all processed Software Packages** *[*UCM shall pro- vide a method RevertProcessedSwPackages to revert all changes done with Pro- cessSwPackage.*♩(RS\_UCM\_00008)*

The main difference between a RevertProcessedSwPackages and a Rollback is that the former can only be performed before the successful activation of the targeted Software Package(s) while the latter can only be performed after such activation.

Depending on the capabilities of UCM and of the updated target, RevertPro- cessedSwPackages is used to revert all the changes that have been applied by ProcessSwPackage. Cancel is also used to revert the changes of the Software Package for which processing started by ProcessSwPackage method call and iden- tified by TransferId. For example, if an application with large resource files is updated “in place” (i.e. in the same partition) then it might not be feasible to revert the update. In this case, to perform a rollback the triggering application could download a Software Package to restore a stable version of the application.

### Error handling during Processing Software Packages

**[SWS\_UCM\_00277]**{DRAFT} **ProcessSwPackage error handling order** *[*Pro- cessSwPackage method shall check the following error conditions and return the re- spective error code.

1. [[SWS\_UCM\_00219](#_bookmark67)]
2. [[SWS\_UCM\_00017](#_bookmark68)]
3. [[SWS\_UCM\_00218](#_bookmark69)]
4. [[SWS\_UCM\_00098](#_bookmark54)]
5. [[SWS\_UCM\_00161](#_bookmark70)]
6. [[SWS\_UCM\_00029](#_bookmark71)]
7. [[SWS\_UCM\_00285](#_bookmark72)]
8. [[SWS\_UCM\_00231](#_bookmark73)]
9. [[SWS\_UCM\_00217](#_bookmark74)]
10. [[SWS\_UCM\_00267](#_bookmark75)]
11. [[SWS\_UCM\_00104](#_bookmark76)]
12. [[SWS\_UCM\_00245](#_bookmark33)]
13. [[SWS\_UCM\_00103](#_bookmark102)]
14. [[SWS\_UCM\_00150](#_bookmark77)]

*♩(RS\_UCM\_00026)*

**[SWS\_UCM\_00219] ProcessSwPackage OperationNotPermitted** *[*ProcessSw- Package shall raise the error ApplicationError OperationNotPermitted in case the processing of the specified Software Package is already done.*♩(RS\_- UCM\_00025, RS\_UCM\_00026)*

**[SWS\_UCM\_00017] Sequential Software Package Processing** *[*Once method ProcessSwPackage has been called by a client, further calls to the same method shall be rejected with ApplicationError ServiceBusy as long as CurrentSta- tus is different than kProcessing.*♩(RS\_UCM\_00001, RS\_UCM\_00003, RS\_UCM\_- 00026)*

When UCM Master or OTA Client calls ProcessSwPackage method and it raises ApplicationError ServiceBusy, UCM Master can retry the ProcessSwPack- age method again later for the same package. This behaviour is configured by Ucm- RetryStrategy.

**[SWS\_UCM\_00297]**{DRAFT} **Retry Strategy for ServiceBusy** *[*When ProcessS- wPackage returns ApplicationError ServiceBusy more than maximumNum- berOfRetries within retryIntervalTime, then UCM Master shall cancel the ac- tive campaign by transitioning to kCancelling state and delete the failing package.*♩ (RS\_UCM\_00001, RS\_UCM\_00003, RS\_UCM\_00026)*

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

**[SWS\_UCM\_00218] ProcessSwPackage InvalidTransferId** *[*ProcessSwPackage shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId is sent by the client.*♩(RS\_UCM\_00026)*

ProcessSwPackage checks authentication in accordance to [[SWS\_UCM\_00098](#_bookmark54)] (

AuthenticationFailed)

**[SWS\_UCM\_00161] Check Software Package version compatibility against UCM version** *[*At ProcessSwPackage, TransferData or TransferExit calls, UCM shall raise ApplicationError IncompatiblePackageVersion if the version for the Soft- ware Package expressed by minimumSupportedUcmVersion attribute is higher than the current version of UCM as available in version attribute.*♩(RS\_UCM\_00007)*

The Software Package is generated by a tooling including a packager which version could not match with the UCM version, leading to manifest interpretation issues for instance.

**[SWS\_UCM\_00029] Consistency Check of Manifest** *[*UCM shall validate the content of the manifest against the schema defined for the meta-data(eg: for missing parameter or for value out of range of the parameter) and shall raise the ApplicationError InvalidPackageManifest if it finds discrepancies there.*♩(RS\_UCM\_00012)*

**[SWS\_UCM\_00285]**{DRAFT} **Removing or updating a Software Cluster not ex- isting in the Machine** *[*If a Software Package’s action is to remove or update a

Software Cluster that is not at one of the states kPresent, kRemoved, kUpdat- ing and kAdded, UCM shall raise ApplicationError SoftwareClusterMissing when ProcessSwPackage is called.*♩(RS\_UCM\_00015)*

**[SWS\_UCM\_00231]**{DRAFT} **ProcessSwPackage IncompatibleDelta** *[*ProcessSwPackage shall raise the error ApplicationError Incompati- bleDelta if deltaPackageApplicableVersion is different from the currently installed version of the referenced SoftwareCluster.*♩(RS\_UCM\_00007)*

**[SWS\_UCM\_00217]**{DRAFT} **ProcessSwPackage InsufficientMemory** *[*Pro- cessSwPackage method shall raise the ApplicationError InsufficientMem- ory if the UCM buffer has not enough resources to process the corresponding Soft- ware Package.*♩(RS\_UCM\_00013, RS\_UCM\_00025)*

**[SWS\_UCM\_00267]**{DRAFT} **Error when checksum is not recognised at process- ing time** *[*If checksum attribute of ArtifactChecksum or CryptoProvider are not recognised, UCM shall raise the ApplicationError InvalidChecksumDescrip- tion.*♩(RS\_UCM\_00012)*

**[SWS\_UCM\_00104] Integrity Check of processed Package** *[*UCM shall raise the ApplicationError ProcessedSoftwarePackageInconsistent if integrity check of the processed Software Packages fails.*♩(RS\_UCM\_00012)*

This operation is realized by the UCM to verify that it did not corrupt any files during the processing. This integrity check is vendor specific and may be realized by the UCM by checking the payload Checksum or by any other mechanisms.

ProcessSwPackage checks if the Software Cluster to be removed is from cat- egory PLATFORM\_CORE. If this is the case, UCM shall not remove it in accordance to [[SWS\_UCM\_00245](#_bookmark33)].

ProcessSwPackage checks if the Software Cluster version being updated is older than currently present in Machine in accordance to [[SWS\_UCM\_00103](#_bookmark102)] (Old- Version).

**[SWS\_UCM\_00150] Cancellation of a Software Package processing** *[*ProcessS- wPackage method shall raise the ApplicationError ProcessSwPackageCan- celled if the Cancel method has been called during the processing of a Software Package.*♩(RS\_UCM\_00024)*

### Error handling for Cancel

**[SWS\_UCM\_00278]**{DRAFT} **Cancel error handling order** *[*Cancel method shall check the following error conditions and return the respective error code.

1. [[SWS\_UCM\_00234](#_bookmark79)]
2. [[SWS\_UCM\_00235](#_bookmark80)]

*♩(RS\_UCM\_00020)*

**[SWS\_UCM\_00234] Cancel OperationNotPermitted** *[*Cancel shall raise the error ApplicationError OperationNotPermitted in case the targeted Software Package processing has not yet started or has been already finished.*♩(RS\_UCM\_- 00020)*

**[SWS\_UCM\_00235] Cancel InvalidTransferId** *[*Cancel shall raise the error Appli- cationError InvalidTransferId in case an invalid TransferId is sent by the client.*♩(RS\_UCM\_00020)*

### Error handling for RevertProcessedSwPackages [SWS\_UCM\_00279]{DRAFT} RevertProcessedSwPackages error handling order

*[*RevertProcessedSwPackages method shall check the following error conditions

and return the respective error code.

1. [[SWS\_UCM\_00237](#_bookmark82)]
2. [[SWS\_UCM\_00236](#_bookmark83)]

*♩(RS\_UCM\_00020)*

**[SWS\_UCM\_00237] RevertProcessedSwPackages OperationNotPermitted** *[*Re- vertProcessedSwPackages method call shall raise the error ApplicationError OperationNotPermitted in case the processed Software Packages are suc- cessfully activated or it is called at other states than kReady (Software Package(s) are finished being processed) or kProcessing states.*♩(RS\_UCM\_00020)*

**[SWS\_UCM\_00236]**{DRAFT} **RevertProcessedSwPackages NotAbleToRevert- Packages** *[*RevertProcessedSwPackages shall raise the error Application- Error NotAbleToRevertPackages in case reverting of processed Software Packages have failed.*♩(RS\_UCM\_00020)*

### Error handling for GetSwProcessProgress

**[SWS\_UCM\_00220] GetSwProcessProgress InvalidTransferId** *[*GetSwPro- cessProgress shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId is sent by the client.*♩(RS\_UCM\_00023)*

### Activation and Rollback

UCM should notify the activation or rollback of Software Packages to other Func- tional Clusters of the AUTOSAR Adaptive Platform. Vendor specific solution dictates to which modules this information is available, in which form and if this is done directly when change is done or when change is executed.

### Activation

The SoftwareCluster state kPresent does not express whether a Soft- wareCluster is currently executed or not. You can refer to chapter [7.1.1](#_bookmark28) Software Cluster Lifecycle for more details about kPresent state and sequence diagram 10.4 for more details about activation.

An activation of SoftwareClusters is triggered by an Activate method call. At be- ginning of activation, UCM is asking State Management for an update session. Once granted, UCM is requesting State Management to stop running processes from the outdated SoftwareClusters. When processes stopped, UCM makes available to the AUTOSAR Adaptive Platform the updated or installed SoftwareClusters, the core action step of the activation. A verification of the activated SoftwareClusters is then performed by requesting State Management changing the SoftwareClus- ters Function Groups modes to kVerify. For an example of activation sequence, you can refer to chapter 10.4

**[SWS\_UCM\_00293]**{DRAFT} **VerifyUpdate method** *[*At kVerifying state and before triggering to kActivated state, UCM shall call the State Management Up- dateRequest Service Interface VerifyUpdate method passing the list of Func- tion Groups defined in SoftwareCluster claimedFunctionGroup attribute of the class.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00107] Activated state** *[*UCM state kActivated shall be set after the new versions of updated SoftwareClusters have been verified.*♩(RS\_UCM\_00008, RS\_UCM\_00030)*

The state management [[3](#_bookmark5)] on the level of execution is handled by the UCM’s client controlling the update process.

UCM has to be able to update several SoftwareClusters for an update campaign. However, these SoftwareClusters could have dependencies not satisfied if updates are processed and activated one by one. Therefore, UCM splits the activation action from the general package processing.

**[SWS\_UCM\_00027]**{OBSOLETE} **Delta Package version applicability** *[*Applicable version of SoftwareCluster on which to apply delta shall be included into related SoftwarePackage’s deltaPackageApplicableVersion attribute.*♩(RS\_UCM\_- 00007)*

Applicable version of a SoftwareCluster on which to apply delta is included into related SoftwarePackage’s deltaPackageApplicableVersion attribute

**[SWS\_UCM\_00025]**{OBSOLETE} **Activation of SoftwareClusters** *[*At the invoca- tion of method Activate, UCM shall enable execution of any pending changes from the previously processed Software Packages.*♩(RS\_UCM\_00021)*

Every call to ProcessSwPackage makes necessary preparations of possible actions on the Software Cluster (ActionType [SWS\_UCM\_00132]) : kInstall, kRe- move, kUpdate. The Activate call finalises the started actions during processing

and then UCM applies changes at activation that were still pending from processing, like for instance updating the list of processes managed by Execution Management.

After Activate, the new set of SoftwareClusters can be started. Activation covers all the processed Software Packages for all the clients.

**[SWS\_UCM\_00022] Activation of Software Clusters** *[*UCM shall activate all the Software Clusters extracted from the Software Packages when Activate is called.*♩(RS\_UCM\_00021)*

The activation method could lead to a full system reset. When Software Package updates underlying OS, AUTOSAR Adaptive Platform or any Adaptive Appli- cation which is configured to be part of Function Group MachineFG, the execu- tion of updated software occurs through system reset by calling State Management UpdateRequest Service Interface ResetMachine method. Meta-data of Software Package defines the activation method.

In principle, it is possible to activate multiple versions of the same SoftwareCluster in one activation step. This could be useful for example with delta package updates but does not apply to firmware updates. The specification does not prohibit to create this kind of chained updates. The decision to use chained updates should be based on safety aspects and the applicability of the underlying update technology, if the update is for a classic or an adaptive platform, if a file system is involved or if the used platform even support it.

### Error handling for Activate

**[SWS\_UCM\_00281] Activate error handling order** *[*Activate method shall check the following error conditions and return the respective error code.

1. [[SWS\_UCM\_00241](#_bookmark87)]
2. [[SWS\_UCM\_00026](#_bookmark88)]
3. [[SWS\_UCM\_00258](#_bookmark39)]
4. [[SWS\_UCM\_00242](#_bookmark89)]
5. [[SWS\_UCM\_00280](#_bookmark90)]

*♩(RS\_UCM\_00026)*

**[SWS\_UCM\_00241] Activate OperationNotPermitted** *[*Activate shall raise the er- ror ApplicationError OperationNotPermitted in case the UCM state is not kReady.*♩(RS\_UCM\_00021)*

**[SWS\_UCM\_00026] Dependency Check** *[*During the UCM state kActivating, UCM shall perform a dependency check to ensure that all the Software Clusters hav- ing dependencies are not missing any necessary Software Cluster as defined by

dependsOn and do not conflict towards each other as defined by conflictsTo, oth- erwise return ApplicationError MissingDependencies.*♩(RS\_UCM\_00007)*

If Activate method cannot establish an Update Session with State Management, it returns UpdateSessionRejected, see [[SWS\_UCM\_00258](#_bookmark39)].

When UCM Master or OTA Client calls Activate method and it raises Appli- cationError UpdateSessionRejected, UCM Master can retry the Activate method again later to enter Update Session again. This behaviour is configured by UcmRetryStrategy.

**[SWS\_UCM\_00298]**{DRAFT} **Retry Strategy for UpdateSessionRejected** *[*When Activate returns ApplicationError UpdateSessionRejected more than maximumNumberOfRetries within retryIntervalTime, then UCM Master shall cancel the active campaign by transitioning to kCancelling state.*♩(RS\_UCM\_00026, RS\_UCM\_00024)*

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

**[SWS\_UCM\_00242] Activate PrepareUpdateFailed** *[*Activate shall raise the er- ror ApplicationError PrepareUpdateFailed in case of activation state transi- tion failure from State Management side.*♩(RS\_SM\_00001)*

**[SWS\_UCM\_00280] Activate VerificationFailed** *[*Activate shall raise the error ApplicationError VerificationFailed in case of verification failure returned by State Management.*♩(RS\_UCM\_00021)*

### Rollback

**[SWS\_UCM\_00005]**{OBSOLETE} **Rollback to the software prior to Finish the up- date process** *[*UCM shall provide a method Rollback to recover from an activation that went wrong.*♩(RS\_UCM\_00008)*

Rollback can be called in the case of A/B partitions or UCM uses some other solution to maintain backups of updated or removed Software Packages.

**[SWS\_UCM\_00110] Rolling-back the software update** *[*At kRollingBack state, UCM shall disable the changes done by the software update by calling State Manage- ment UpdateRequest Service Interface PrepareRollback method for each Func- tion Group of the processed Software Cluster in the update session. Then UCM shall call State Management UpdateRequest Service Interface ResetMa- chine method if any Software Cluster requires a machine reboot to be rolled back.*♩(RS\_UCM\_00008)*

If a reset of the Machine is not necessary, an implementation specific way to in- form Execution Management that a Software Cluster was updated can be per- formed.

**[SWS\_UCM\_00299]**{DRAFT} **Verify rolled back Software Clusters** *[*After a UCM successful Rollback using call State Management UpdateRequest Service In- terface PrepareRollback method and optional Machine reset or manifest reparse, UCM shall call State Management UpdateRequest Service Interface VerifyUp- date method to confirm that all Software Clusters impacted by update are still safe to be launched.*♩(RS\_UCM\_00008)*

**[SWS\_UCM\_00302]**{DRAFT} **Rollback failing is triggering production error** *[*When a Rollback is failing, UCM shall report UCM\_FAILED\_ROLLBACK production error.*♩ (RS\_UCM\_00008, RS\_UCM\_00027)*

### Error handling for Rollback

**[SWS\_UCM\_00282] Rollback error handling order** *[*Rollback method shall check the following error conditions and return the respective error code.

1. [[SWS\_UCM\_00239](#_bookmark92)]

*♩(RS\_UCM\_00008)*

**[SWS\_UCM\_00239] Rollback OperationNotPermitted** *[*Rollback shall raise the er- ror ApplicationError OperationNotPermitted in case UCM current state is not kActivated nor kVerifying.*♩(RS\_UCM\_00020)*

### Boot options

During update process the executed software is switched from original software to updated software and in case of rollback, from updated software to original version. Which version of software is executed is dependent on the UCM state and this is man- aged by the UCM. In case of platform and OS update the switch between software versions occurs through system reset and depending on the system design the Exe- cution Management [[2](#_bookmark4)] might be started before UCM. In this case there can’t be direct interface between UCM and Execution Management [[2](#_bookmark4)] to define which versions of soft- ware would be executed. Instead this would be controlled through persistent controls which are referred as Boot options in this document.

**[SWS\_UCM\_00094] Management of executable software** *[*UCM shall manage which version of software is available for the Execution Management [[2](#_bookmark4)] to launch.*♩(RS\_- UCM\_00021)*

During the kActivating state, UCM modifies the Boot options so that in the next restart for the updated software the new versions will be executed. In the kRolling- Back state, UCM modifies the Boot options so that in the next restart of the updated software the original versions will be executed.

### Finishing activation

**[SWS\_UCM\_00020] Finishing the packages activation** *[*UCM shall provide a method Finish to commit all the changes and clean up all temporary data of the processed Software Packages.*♩(RS\_UCM\_00015)*

UCM should also remove Software Packages, logs or any older versions of changed software to save storage space. It is up to implementer to remove or not the Software Packages.

**[SWS\_UCM\_00259] Ending the update session** *[*UCM shall call State Manage- ment UpdateRequest Service Interface StopUpdateSession method when UCM is exiting the kCleaningUp state.*♩(RS\_UCM\_00021, RS\_UCM\_00018)*

**[SWS\_UCM\_00240] Finish OperationNotPermitted** *[*Finish shall raise the error ApplicationError OperationNotPermitted in case there are no activated nor rolled-back Software Packages pending finalization (i.e UCM state is not kActi- vated nor kRolledBack.*♩(RS\_UCM\_00001, RS\_UCM\_00026)*

For UCM to be able to free all unneeded resources while processing the Finish re- quest, it is up to the vendor and platform specific implementation to make sure that obsolete versions of changed SoftwareClusters aren’t executed anymore.

### Status Reporting

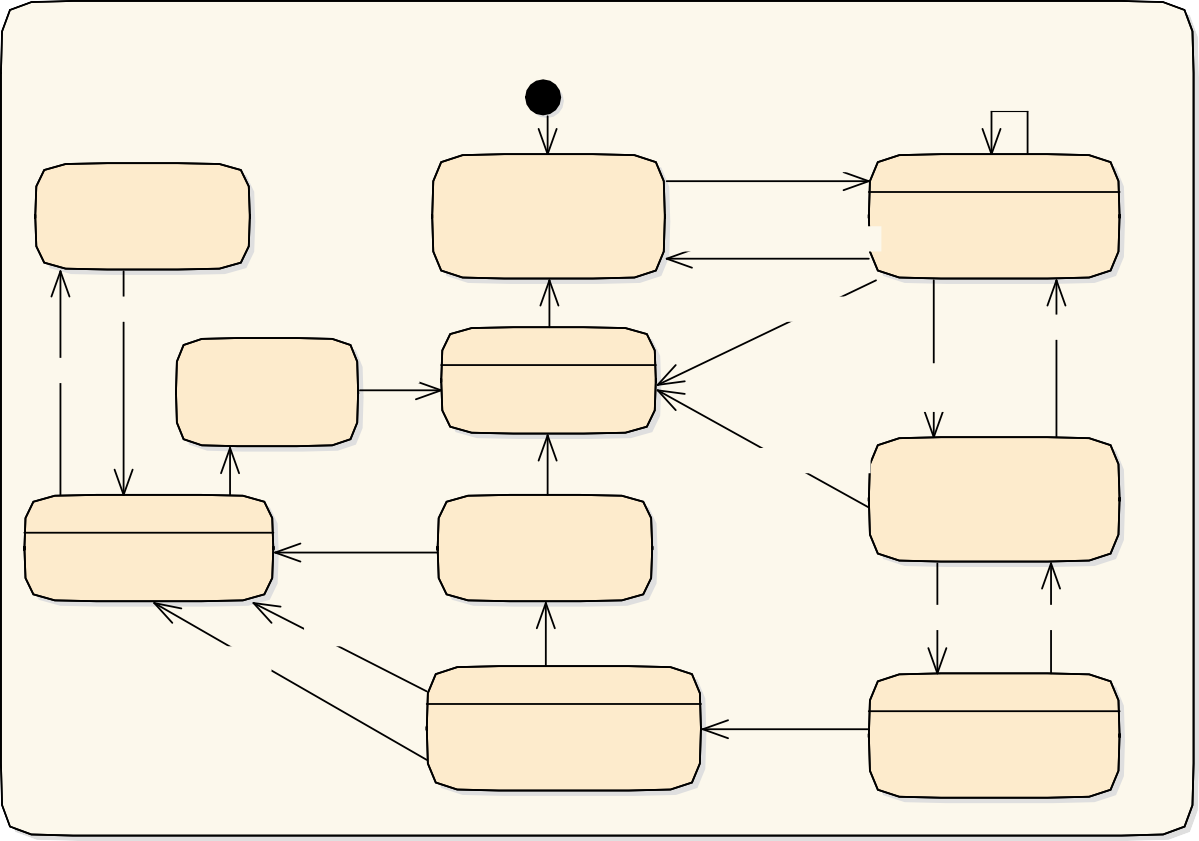
Once Software Packages are transferred to UCM, they are ready to be processed to finally apply changes to the AUTOSAR Adaptive Platform. In contrast to the transmission, the processing and activation tasks have to happen in a strict sequential order.

To give an overview of the update sequence, the global state of UCM is described in this section. The details of the processing and activation phases and the methods are specified in the [7.1.5](#_bookmark65) and [7.1.6](#_bookmark85).

The global state of UCM can be queried using the field CurrentStatus. The state machine for CurrentStatus is shown in Fig. [7.5](#_bookmark96). This diagram does not include behaviour after a reset. Examples can be found of how UCM and its CurrentStatus field behave including reset management in chapter 10 Sequence Diagram.

**[SWS\_UCM\_00019] Status Field of Package Management** *[*The global state of UCM

shall be provided using the field CurrentStatus*♩(RS\_UCM\_00024)*



**Package ManagerStatus**

Initi al

Cancel

ProcessSw Package

**kRollingBackFailed**

**kIdle**

**kProcessing**

Cancelled do / process SW package [ProcessedPackages=0]

Rollback

RevertProcessedSw Packages

ProcessSw Package

**kCleaningUp**

OnFailure

**kRolledBack**

Finish

do / clean up

Cancelled,

PackageProcessed [ProcessedPackages>0]

**kReady**

O nSuccess

RevertProcessedSw Packages

**kRollingBack**

Finish

**kActivated**

Rollback

do / roll back to old version

Activate OnFailure

Rollback

OnFailure

O nSuccess

**kVerifying**

**kActivating**

do / MonitorApplicationRestart

O nSuccess

do / DependencyCheck

**Figure 7.5:** **State Machine for the package processing using service interface: Package- Management**

UCM supported method calls for each value of field CurrentStatus are shown in Fig. [7.5](#_bookmark96).

**[SWS\_UCM\_00080] Idle state of Package Management** *[*kIdle shall be the default state.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00149] Return to the Idle state from Processing state** *[*kIdle state shall be set when ProcessSwPackage returns with error code ProcessSwPackage- Cancelled and if no other Software Packages were previously processed during this processing operation.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00151] Entering the Ready state of Package Management after a Cancel call** *[*If ProcessSwPackage has been cancelled, UCM shall return error code ProcessSwPackageCancelled and set state to kReady only if at least one other Software Package was previously processed during this processing operation.*♩ (RS\_UCM\_00024)*

**[SWS\_UCM\_00081] Processing state of Package Management** *[*kProcessing state shall be set only if ProcessSwPackage has been called. This shall only be possible, if CurrentStatus is reported as kIdle or kReady.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00266]**{DRAFT} **OperationNotPermitted error and UCM state** *[*UCM shall return ApplicationError OperationNotPermitted if ProcessSwPack- age is called by a client with UCM at CurrentStatus state different than kIdle, kProcessing or kReady.*♩(RS\_UCM\_00001, RS\_UCM\_00004)*

**[SWS\_UCM\_00083] Entering the Ready state of Package Management after a suc- cessful processing operation** *[*kReady state shall be set after a Software Pack- age processing has been completed successfully.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00265] state transition due to ProcessSwPackage error** *[*If Pro- cessSwPackage raises an ApplicationError other than ProcessSwPackage- Cancelled, it shall transition from kProcessing to kIdle if no other Software Pack- ages were previously processed during this processing operation, or kReady if at least one other Software Package was previously processed before the failed processing operation, and shall perform clean-up actions.*♩(RS\_UCM\_00015, RS\_UCM\_00026)*

Clean-up actions could be similar to the cancel call by for instance deleting files, folders or artefacts of the processed Software Cluster.

**[SWS\_UCM\_00152] Entering the Ready state of Package Management after a missing dependency** *[*kReady state shall be set when Activate fails due to an ApplicationError MissingDependencies.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00084] Entering the kActivating state of Package Management** *[*kAc- tivating shall be set when Activate is called. This triggers the dependency check and returns ApplicationError MissingDependencies if this check fails.*♩(RS\_- UCM\_00024)*

**[SWS\_UCM\_00153] Action in kActivating state of Package Management** *[*When kActivating is set and after the State Management UpdateRequest Service In- terface RequestUpdateSession method call by UCM, the UCM shall call the State Management UpdateRequest Service Interface PrepareUpdate method for the concerned Software Cluster including a list of all Function Groups belonging to that Software Cluster.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00260]**{DRAFT} **PrepareUpdate, VerifyUpdate and PrepareRollback orders** *[*UCM shall compute the order of the State Management UpdateRequest Service Interface PrepareUpdate, VerifyUpdate and PrepareRollback method calls from the dependency model included in the Software Cluster manifests.*♩ (RS\_UCM\_00007, RS\_UCM\_00021, RS\_UCM\_00030)*

**[SWS\_UCM\_00261] PrepareUpdate, VerifyUpdate and PrepareRollback syn- chronous calls** *[*Calls to State Management UpdateRequest Service Interface PrepareUpdate, VerifyUpdate and PrepareRollback methods shall not be concurrent.*♩(RS\_UCM\_00026)*

**[SWS\_UCM\_00262]**{DRAFT} **Update preparation rejected** *[*If any call of the State Management UpdateRequest Service Interface PrepareUpdate method returns error kRejected too many times (maxNumberOfPrepareUpdateRetries) or for

too long (prepareUpdateTimeout), UCM shall transition from kActivating to

kReady states.*♩(RS\_UCM\_00026)*

**[SWS\_UCM\_00263] Update preparation failure** *[*If any one of the State Man- agement UpdateRequest Service Interface PrepareUpdate method returns error kFailed, UCM shall transition from kActivating to kReady states.*♩(RS\_UCM\_- 00026)*

**[SWS\_UCM\_00154] Entering the Verifying state of Package Management** *[*kVer- ifying shall be set when the dependency check have been performed successfully (all dependencies are satisfied) and that the preparation of the Software Clusters by the State Management has been successfully performed.*♩(RS\_UCM\_00024)*

The machine could most likely be restarted in case a A/B partition is used. In case the A/B partition is not used, all affected Function Groups or the platform could be restarted. Immediately after the processed Software Package has been restarted, a system check has to be performed in order to make sure the machine is able to start up as expected. With this check it is verified that other safety relevant software like Functional Cluster Platform Health Manager [[13](#_bookmark13)] is running and user can be protected from any issues caused by the update after the update has finished.

An update could most likely require to reparse the manifests after performing the atomic activation of the Software Clusters (switching A/B partition, changing sym- links, etc.) if a machine reset is not needed.

**[SWS\_UCM\_00085] Entering the kActivated state of Package Management** *[*kAc- tivated state shall be set when the VerifyUpdate method of State Management service interface UpdateRequest is returned successfully.*♩(RS\_UCM\_00024)*

By a successful return of VerifyUpdate, UCM assumes all impacted Function Groups (the ones related to the processed Software Package) have been suc- cessfully restarted and verified.

kVerifying state gives the client controlling the update process a chance to perform verification test by calling State Management UpdateRequest Service Interface [SWS\_SM\_91017] VerifyUpdate method, though functionality in verify state can be limited. Client can also coordinate the results over several AUTOSAR Adaptive Platforms and still perform a Rollback if verification indicates the need for it.

If the system check is successful, the client can decide either to Rollback the current active processing so that the previous processed working software gets started, or to perform Finish so that the changes of processed software become permanent. By calling Finish a clean-up is initiated and in case of A/B partition, a swap between the partitions happens and the newly inactive partition becomes a copy of the newly active partition. In case Finish succeeds (including the clean-up), the current Cur- rentStatus changes to kIdle.

For Rollback the update software needs to be deactivated and possibly reactivated from original version, e.g. self-update of UCM. For this reason Rollback is also performed through two states, similarly as activation. Calling Rollback sets UCM

into kRollingBack state where original software version is made executable and where original software is activated by the State Management. This is started by calling State Management UpdateRequest Service Interface [SWS\_SM\_91017] PrepareRollback method for each Software Cluster. On success, UCM goes to kRollingBack state. In this state all the changes introduced during update process have been deactivated and can be cleaned by calling Finish.

**[SWS\_UCM\_00126] Entering the kRollingBack state after a Rollback call** *[*The state kRollingBack shall be set when Rollback is called.*♩(RS\_UCM\_00008, RS\_- UCM\_00030)*

**[SWS\_UCM\_00155] Entering the kRolling-Back state after a failure in the kVeri- fying state** *[*The state kRollingBack shall be set if any of the State Management UpdateRequest Service Interface VerifyUpdate method calls returns the result kFailed.*♩(RS\_UCM\_00008, RS\_UCM\_00030)*

**[SWS\_UCM\_00264]**{DRAFT} **Update verification rejected** *[*If any call of the State Management UpdateRequest Service Interface VerifyUpdate returns error kRe- jected too many times (maxNumberOfVerifyUpdateRetries) or for too long ( verifyUpdateTimeout), UCM shall transition to kRollingBack state.*♩(RS\_UCM\_- 00030, RS\_UCM\_00008)*

**[SWS\_UCM\_00111] Entering the kRollingBack state** *[*The state kRollingBack shall be set after all calls to State Management UpdateRequest Service Interface PrepareRollback have returned successfully.*♩(RS\_UCM\_00008, RS\_UCM\_00030)*

**[SWS\_UCM\_00300]**{DRAFT} **Software Cluster failing to rollback** *[*If Rollback is failing, UCM CurrentStatus shall transition from kRollingBack to kRolling- BackFailed.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00301]**{DRAFT} **Retry ro Rollback again when UCM is in kRolling- BackFailed state** *[*If Rollback method is called while being at kRollingBack- Failed, UCM CurrentStatus shall transition from kRollingBackFailed to kRollingBack.*♩(RS\_UCM\_00024)*

**[SWS\_UCM\_00146] Entering the Cleaning-up state after a Finish call** *[*The state kCleaningUp shall be set when Finish is called and the UCM starts to perform cleanup actions.*♩(RS\_UCM\_00008, RS\_UCM\_00030)*

**[SWS\_UCM\_00162] Entering the Cleaning-up state after a RevertProcessedSw- Packages call** *[*The state kCleaningUp shall be set when RevertProcessedSw- Packages is called in kProcessing or kReady states and the UCM starts to perform cleanup actions.*♩(RS\_UCM\_00008, RS\_UCM\_00030)*

**[SWS\_UCM\_00163] Action in Cleaning-up state** *[*When kCleaningUp state is set, the UCM shall clean up all data of the processed packages that are not needed any- more.*♩(RS\_UCM\_00008, RS\_UCM\_00030)*

**[SWS\_UCM\_00164] Cleaning up of Software Packages** *[*In kCleaningUp state, the

UCM may remove (from the UCM buffer for instance) the "physical" Software Package

(e.g. zip file) that was used to transport the the SoftwareCluster to the UCM.*♩(RS\_- UCM\_00008, RS\_UCM\_00030)*

**[SWS\_UCM\_00127] Finishing update sequence** *[*kIdle shall be set when Finish is called and the clean-up has been successfully performed. This finishes the update sequence and next sequence can be started.*♩(RS\_UCM\_00008, RS\_UCM\_00030)*

**[SWS\_UCM\_00147] Return to the Idle state from Cleaning-up state** *[*kIdle state shall be set when the Clean-up operation has been completed successfully.*♩(RS\_- UCM\_00024)*

### Robustness against reset

Failure during over-the-air updates could lead into corrupted or inconsistent software configuration and further updates might be blocked. For this reason UCM needs to be robust against interruptions like power downs.

**[SWS\_UCM\_00157] Detection of reset** *[*At start up UCM shall identify if uncontrolled reset occurred.*♩(RS\_UCM\_00027)*

The way for UCM to detect uncontrolled reset is project specific. UCM could use hard- ware platform specific registers to detect Soft/Hard reset. Or it could access PHM Functional Cluster to detect uncontrolled reset. UCM could also check that the Cur- rentStatus persistent field is not kIdle or kVerifying.

**[SWS\_UCM\_00158] Cleanup of interrupted actions** *[*After an uncontrolled reset, UCM shall check non volatile memory integrity, recover processed artifacts in case it is corrupted and resume interrupted actions in order to return the system into a state from where UCM can continue serving its Clients.*♩(RS\_UCM\_00027)*

After an uncontrolled reset, it can be possible as an example for UCM to confirm consis- tency of any processed artifacts based on ArtifactChecksum class associated to SoftwareCluster. If checksum value of an artifact does not match, it can be deleted and processed again.

**[SWS\_UCM\_00270] UCM internal state persistency** *[*UCM shall persist Cur- rentStatus state field to be able to resume on-going update after an intended or unintended reboot.*♩(RS\_UCM\_00027)*

### Boot monitoring

Activation failure during OS and Platform-self updates can lead to a state in which the system is not able to reach a point where UCM and the client are able to function as expected and thus not able to execute the rollback. For these cases the system should include component which is responsible to monitor that the OS and platform will start up correctly. In case of failure, the Boot monitoring component should trigger a reset or modify the boot options to trigger a rollback.

### History

**[SWS\_UCM\_00115] History** *[*GetHistory method shall retrieve all actions that have been performed by UCM within a specific time window input parameter.*♩(RS\_UCM\_- 00032)*

In the case the UCM Client requests a rollback after a successful activation, Cur- rentStatus field transitioning to kActivated, GetHistory method will later re- turn GetHistoryType, with subelement Resolution of type ResultType equal to kActivatedAndRolledBack.

**[SWS\_UCM\_00292] History elements ordering** *[*UCM shall return from GetHistory method a vector of GetHistoryType sorted in an increasing chronological order.*♩ (RS\_UCM\_00032)*

**[SWS\_UCM\_00160] Processing results records** *[*When UCM is entering kVeri- fying, UCM shall save activation time based on timeBaseResource and activation result of processed Software Packages in the history.*♩(RS\_UCM\_00032)*

**[SWS\_UCM\_00271]**{DRAFT} **Keeping history of failure error code** *[*UCM shall keep in GetHistoryType subelement FailureError the last failure error code as de- scribed in [SWS\_UCM\_00136]. If no error occurred, the stored value shall be 0.*♩ (RS\_UCM\_00032)*

**[SWS\_UCM\_00303]**{DRAFT} **failing to record history** *[*If UCM is failing to record a new entry in history, UCM shall report a production error: UCM\_HISTORY\_RECORD\_- FAILED. Any successful history update shall report a pass to this production error.*♩ (RS\_UCM\_00032)*

### Version Reporting

**[SWS\_UCM\_00004]**{OBSOLETE} **Report software information** *[*UCM shall provide a method GetSwClusterInfo of the interface service PackageManagement to pro- vide the identifiers and versions of the SoftwareClusters that are in state kPre- sent.*♩(RS\_UCM\_00002)*

**[SWS\_UCM\_00030] Report changes** *[*UCM shall provide a method GetSwCluster- ChangeInfo of the interface service PackageManagement to provide the identifiers and versions of the SoftwareCluster that are in state kAdded, kUpdating or kRemoved.*♩(RS\_UCM\_00011)*

**[SWS\_UCM\_00185] Provide SoftwareCluster general information** *[*At the invo- cation of method GetSwClusterDescription, UCM shall return the version, type approval, license and release notes of the SoftwareCluster that are in state kPre- sent.*♩(RS\_UCM\_00002, RS\_UCM\_00011)*

### Securing Software Updates

UCM provides service interface using ara::com. There is no authentication of the client in UCM’s update sequence.

For authentication of the Software Package, you can refer to [7.1.3](#_bookmark40)

**[SWS\_UCM\_00103]**{DRAFT} **Update to older Software Cluster version than currently present and than previously removed** *[*If the version of a Soft- warePackage (returned by GetSwPackages) is smaller than the version of currently present SoftwareCluster (returned by GetSwClusterInfo), the UCM method TransferExit or TransferData shall raise the ApplicationError Old- Version, log this attempt in history (retrievable from GetHistory) and delete the rejected Software Package.*♩(RS\_UCM\_00031)*

**[SWS\_UCM\_00190] Reinstallation of older Software Cluster version than pre- viously removed** *[*New Software Clusters getting installed shall be compared with the history of all installed Software Clusters to prevent installation of a Soft- ware Cluster with a lower or equal version than previously installed.*♩(RS\_UCM\_- 00003, RS\_UCM\_00031)*

**[SWS\_UCM\_CONSTR\_00002]**{DRAFT} **UCM confidential information handling** *[*The PackageManagement interface shall only be mapped via ara::com to a se- cure endpoint using secure communication channel providing confidentiality protec- tion.*♩(RS\_UCM\_00002, RS\_UCM\_00010, RS\_UCM\_00011)*

The GetSwClusterInfo, GetSwClusterChangeInfo, GetHistory, GetSwClusterDescrip- tion and GetSwPackages methods are using data that could identify vehicle user and therefore should be protected for confidentiality.

**[SWS\_UCM\_00202]**{DRAFT} **Trusted Platform compliance** *[*UCM shall ensure that after processing updates, all the necessary changes to comply with the Trusted Plat- form are applied.*♩(RS\_EM\_00014)*

The authentication tag of the Trusted Platform corresponding to the updated/re- moved/added executable files should also be updated/removed/added. See also Chap- ter 7.10 of the Execution Management [[2](#_bookmark4)] for details on the Trusted Platform.

### Functional cluster lifecycle

**[SWS\_UCM\_00274]**{DRAFT} **UCM initialization** *[*UCM shall offer its services only af- ter its internal initialization has been completed, after reporting kRunning state to Execution Management.*♩(RS\_UCM\_00044)*

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

### Shutdown behaviour

There are no requirements of shutdown behaviour from UCM functional cluster.

## UCM Master

### UCM Master Functional Cluster lifecycle

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

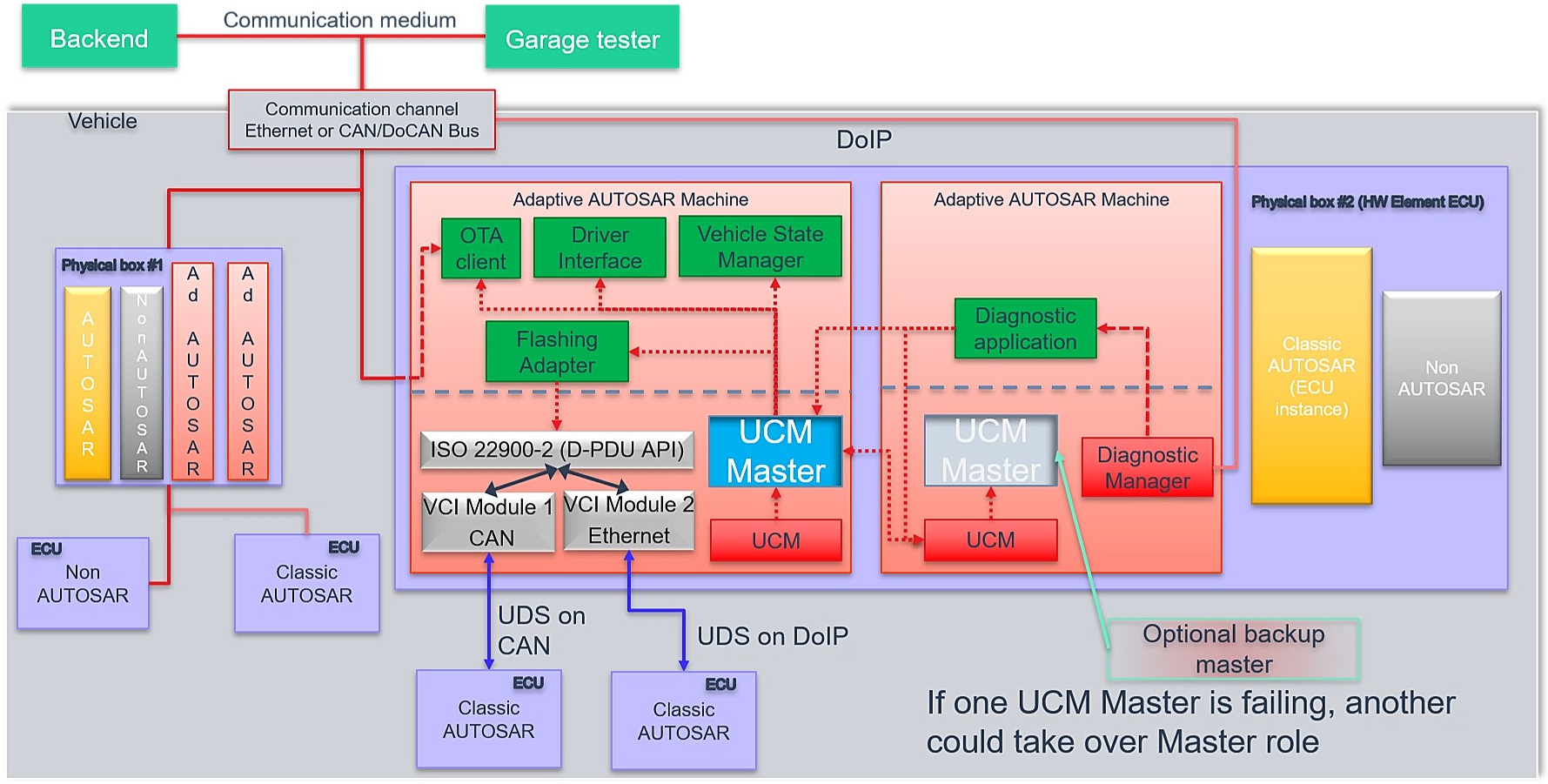
**[SWS\_UCM\_01019]**{DRAFT} **UCM Master initialization** *[*UCM Master shall offer its services only after its internal initialization has been completed, after switching to Run- ning state.*♩(RS\_UCM\_00044)*

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

### Technical Overview

UCM Master 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.

UCM Master receives packages from Backend or Diagnostic tool, parses and inter- prets the Vehicle Package, transfers or streams Software Packages to suitable targets (UCM subordinate or Diagnostic Application) and orchestrates the processing, activations and eventual rollbacks. All these actions are what is called a campaign which UCM Master is coordinating. The UCM of the machines in the same network of a UCM Master, candidates target of a campaign, are referred to as UCM subordinates.



**Figure 7.6: Example of UCM Master architecture overview within a vehicle**

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

The OTA Client establishes a communication between Backend and UCM Master so that they can exchange information of the installed Software Clusters in the vehicle and the Software Clusters available in the Backend. This communica- tion could be triggered by OTA Client with a scheduler and UCM Master to request the updates in case of newly available Software Clusters (pull case) or by Back- end 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 UCM Master 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 finan- cial 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 dur- ing 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.

### UCM Master general behaviour

The UCM Master acts as a client of the service interface offered by the UCM subor- dinates, already specified in UCM. However, the UCM Master also offers three differ- ent service interfaces to OTA Client, Vehicle Driver interface and Vehicle State Manager respectively. UCM Master aggregates UCM subordinates states and can re- port its status field to a Backend through its OTA Client.

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

**[SWS\_UCM\_01003] UCM Master checks states of UCM subordinates** *[*A UCM Mas- ter shall check the status of its UCM subordinates are all at kIdle CurrentStatus state before starting a campaign.*♩(RS\_UCM\_00043)*

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

### UCM identification

For UCM Master to distribute Software Packages to other UCM subordinates, UCM Master has to identify UCM subordinates 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 UCM Master transferring packages to targeted UCMs. To get such identifier, UCM Master will perform first a service discovery through ara::com to get all UCMs service instances available. Then UCM Master will call GetId method for each UCM subordinates returning each corresponding UcmModuleInstantiation identi- fiers.

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

**[SWS\_UCM\_01005] UCM Master is discovering UCMs in vehicle** *[*UCM Master shall continuously look for UCM service instances (use of StartFindService() call).*♩ (RS\_UCM\_00036)*

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

Default (at boot) Master/Subordinate hierarchy or priority could be optionally overwrit- ten for each campaign based on Vehicle Package content at the condition OTA Client could properly parse Vehicle Packages.

### UCM Master Software Packages transfer or streaming

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

It is necessary to distinguish Vehicle Package (UCM Master specific) from Soft- ware Packages transfer.

**[SWS\_UCM\_01011] TransferVehiclePackage InsufficientMemory** *[*Transfer- VehiclePackage method shall raise the ApplicationError InsufficientMem- ory if the UCM buffer has not enough resources to process the corresponding Vehicle Package.*♩(RS\_UCM\_00013)*

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

**[SWS\_UCM\_01014] Packages transferring sequence** *[*TransferStart method shall raise the ApplicationError UnexpectedPackage if the Software Pack- age name parameter was not a value of the RequestedPackage field.*♩(RS\_UCM\_- 00043)*

**[SWS\_UCM\_01013] Too big block size received by UCM Master** *[*In the case the re- ceived block size with TransferData exceeds the block size returned by Transfer- Start or TransferVehiclePackage for the same TransferId, UCM Master shall raise the ApplicationError IncorrectBlockSize.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01015] Invalid Vehicle Package manifest** *[*TransferExit shall raise the ApplicationError InvalidPackageManifest when a Vehicle Package manifest is not compliant with the AUTOSAR schema.*♩(RS\_UCM\_00036, RS\_UCM\_- 00043)*

**[SWS\_UCM\_01016] Invalid Package Manifest** *[*UCM Master shall raise the Appli- cationError InvalidPackageManifest in case a manifest file is not compliant with the AUTOSAR schema.*♩(RS\_UCM\_00036, RS\_UCM\_00043)*

**[SWS\_UCM\_01017] RequestedPackage field** *[*The field RequestedPackage shall contain the requested Software Package name and version as configured in cam- paign which is modelled by VehiclePackage.*♩(RS\_UCM\_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 UCM Master’s RequestedPackage field. Version is added to support campaigns which need an up- date 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.

When UCM Master or OTA Client calls TransferData method and it raises Ap- plicationError BlockInconsistent, UCM Master or OTA Client can retry the TransferData method again later for the same block. This behaviour is config- ured by UcmRetryStrategy.

**[SWS\_UCM\_01020]**{DRAFT} **Retry Strategy for BlockInconsistent** *[*When Trans- ferData returns ApplicationError BlockInconsistent more than maximum- NumberOfRetries within retryIntervalTime, then UCM Master shall cancel the active campaign by transitioning to kCancelling state and delete the failing pack- age.*♩(RS\_UCM\_00012)*

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

### Adaptive Applications interacting with UCM Master

In order to have interoperability between several vendors platforms, Adaptive Ap- plications interacting with UCM Master 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.

### OTA Client

OTA Client is an Adaptive Application that sets communication channel be- tween Backend and UCM Master. 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 UCM Master as a service provider via ara::com. Since trans- ferring Vehicle Packages and Software Packages from Backend to UCM Mas- ter 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 Master Software Packages transfer or streaming as specified in chapter [7.2.5](#_bookmark111), it should then provide at least the following functionality:

* Comply to the requirements of chapter [7.1.3](#_bookmark40) in the context of package transfer between OTA Client and UCM Master.
* OTA Client should subscribe to UCM Master’s RequestedPackage field to know what Software Package is expected to be transferred
* OTA Client should subscribe to UCM Master’s TransferState field to know what is campaign state
* OTA Client should subscribe to UCM Master’s SafetyState field to eventu- ally 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](#_bookmark42)]

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 dif- ferent 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 kTrans- ferring state to allow higher priority campaign to be performed.

Only one UCM Master has to be used by OTA Clients per network domain. As UCM Master 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 UCM Master’s state with TransferState field, in order to avoid any interference and guarantee success of an update campaign.

**[SWS\_UCM\_01101] Provide information of installed Software Clusters in ve- hicle** *[*UCM Master shall provide a method GetSwClusterInfo to return information of all Software Cluster that are in state kPresent.*♩(RS\_UCM\_00033)*

UCM Master 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 UCM Master through OTA Client a Vehicle Package. It is up to OTA Client to make sure the synchroni- sation 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\_UCM\_01103] Inform Backend of needed Software Packages for an up- date** *[*On SwPackageInventory call, UCM Master shall compare the supplied list of available Software Packages in the Backend for the vehicle to its own inter- nal information of present Software Clusters in the vehicle and return the list of Software Packages selected for update.*♩(RS\_UCM\_00033)*

The OTA Client uses this returned Software Packages list to re- quest the selected packages to the Backend. As required by constraint [[SWS\_UCM\_CONSTR\_00014](#_bookmark36)], each Software Cluster corresponds to one Software Package and share the same shortName.

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

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

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

### Vehicle Driver Interface

Vehicle driver interface could be required by legal constrains or communication cost consideration. To support mandatory safety and security critical updates, driver inter- action 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\_UCM\_01105]**{OBSOLETE} **Interaction of UCM Master with Vehicle Driver** *[* UCM Master shall provide a method DriverApproval in order to receive the confir- mation of the vehicle driver’s approval.*♩(RS\_UCM\_00038)*

**[SWS\_UCM\_CONSTR\_00017]**{DRAFT} **Interaction of UCM Master with Vehicle Driver** *[*When vehicle driver accepts or denies the campaign, Vehicle Driver Applica- tion shall call DriverApproval method of VehicleDriverApplicationInter- face to inform UCM Master of the vehicle driver’s decision.*♩(RS\_UCM\_00038)*

The Vehicle Driver Interface Adaptive Application could adapt its notification content related to safety by subscribing to the UCM Master’s SafetyConditions field.

**[SWS\_UCM\_01117]**{DRAFT} **UCM Master SafetyState field** *[*UCM Master shall provide to vehicle driver interface the SafetyConditions field containing the re- quired safety condition for the campaign as configured in safetyCondition.*♩(RS\_- UCM\_00038, RS\_UCM\_00037)*

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

**[SWS\_UCM\_01118] UCM Master waiting for vehicle driver approval** *[*In the case approval from driver is requested as configured in VehiclePackage, UCM Master shall wait for DriverApproval method with parameter Approval=True before tran- sitioning state from kVehiclePackageTransferring to kSoftwarePackage\_- Transferring, kSoftwarePackage\_Transferring to kProcessing or kPro- cessing to kActivating.*♩(RS\_UCM\_00038)*

**[SWS\_UCM\_CONSTR\_00003] Exclusive use of Vehicle Driver Interface** *[*Software Integrator shall ensure that only one Adaptive Application is using the UCM Master’s Vehicle Driver Interface.*♩(RS\_UCM\_00035, RS\_UCM\_00037)*

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

**[SWS\_UCM\_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 SafetyConditions field, the Vehicle Driver Interface Adaptive Application shall call the method ReportUnsupport- edSafetyConditions with parameter UnsupportedSafetyConditions contain- ing all the safety conditions which are not supported by Vehicle Driver Interface Adap- tive Application.*♩(RS\_UCM\_00037)*

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

**[SWS\_UCM\_01135] Get Software Clusters descriptions from a vehicle** *[*At GetSwClusterDescription method call via VehicleDriverApplicationIn- terface interface, UCM Master shall return the descriptions of all Software Clus- ters which are aggregated from all the UCM Subordinates and all Flashing Adapters.*♩ (RS\_UCM\_00033, RS\_UCM\_00038)*

### Vehicle State Manager

Vehicle State Manager is collecting states from the several vehicle ECUs and in- forms UCM Master when the safety state computed based on the safety policy re- ferred in the Vehicle Package is changing. If the safety policy is not met, the UCM Master can for instance decide to:

* Inform vehicle driver that the safety conditions are not met to continue the update
* postpone, pause or cancel the update until policy is met

**[SWS\_UCM\_01109]**{DRAFT} **UCM Master provides a safety interface** *[*UCM Mas- ter shall set the SafetyConditions field taking the value from safetyCondi- tion attribute for each VehicleRolloutStep of the VehiclePackage.*♩(RS\_- UCM\_00037)*

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

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

**[SWS\_UCM\_CONSTR\_00009]**{DRAFT} **Safety condition change** *[*Vehicle State Manager Adaptive Application shall call PublishSafetyState method provided by UCM Master when the field SafetyConditions is changing value.*♩(RS\_UCM\_- 00035, RS\_UCM\_00037)*

**[SWS\_UCM\_CONSTR\_00015]**{DRAFT} **Trigger on kVehicleChecking state** *[*On transition to kVehicleChecking state provided by CampaignState from Ve- hicleStateManagerInterface, Vehicle State Manager shall first perform checks to assess the post-activation state of the vehicle.*♩(RS\_UCM\_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\_UCM\_01272]**{DRAFT} **VehicleCheck call not permitted** *[*UCM Master shall return ApplicationError OperationNotPermitted if VehicleCheck method is called in another UCM Master state than kVehicleChecking.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_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\_UCM\_00035, RS\_UCM\_00037)*

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

**[SWS\_UCM\_01275]**{DRAFT} **Safety conditions during activation** *[*UCM Master shall apply the safetyCondition of the last VehicleRolloutStep to Safety- Conditions field during the ECUs activations as configured in the VehiclePack- age*♩(RS\_UCM\_00043, RS\_UCM\_00037)*

**[SWS\_UCM\_CONSTR\_00007]**{DRAFT} **Unsupported safety conditions by Vehicle State Manager** *[*In the case the requested SafetyConditions field is not referring to an existing safety condition implemented by Vehicle State Manager, the Ve- hicle State Manager shall call VehicleStateManagerInterface service in- terface PublishSafetyState method with parameter SafetyStates containing at least one value equal to ’NotSupported’.*♩(RS\_UCM\_00037)*

**[SWS\_UCM\_CONSTR\_00008]**{DRAFT} **Switching vehicle into update mode** *[*Ve- hicle State Manager shall change vehicle’s state and its ECUs in the right update mode in order to avoid any timeout issues during update.*♩(RS\_UCM\_00037)*

This vehicle state change could be triggered based on UCM Master State Machine.

### Flashing Adapter

Flashing Adapter is an application that is used in the case UCM Master 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 UCM Master and the AUTOSAR Adaptive Platform, and uses an implemen- tation of diagnostic protocol data unit application programming interface (D-PDU API) to communicate with Classic ECUs over the Vehicle Bus.

The data transfer from Flashing Adapter to the target ECU via diagnostic communi- cation 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 spe- cific 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\_UCM\_CONSTR\_00011]**{DRAFT} **Flashing Adapter provided interface** *[*Flashing Adapter shall provide the same ara::com service interface as UCM ([SWS\_UCM\_00131]).*♩(RS\_UCM\_00035)*

### Non Adaptive Platform update

**[SWS\_UCM\_01121]**{DRAFT} **Adaptive Platform interface provided for Flashing Adapter** *[*The interface provided by the AUTOSAR Adaptive Platform in order to update non AUTOSAR Platform should comply with ISO 22900-2:2017 (D-PDU API) but 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.*♩(RS\_UCM\_00035)*

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.

### D-PDU API implementation support

**[SWS\_UCM\_01122]**{DRAFT} **Supported physical layers by D-PDU API imple- mentation** *[*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, all other physical layers present in D-PDU API are optional.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01123]**{DRAFT} **Supported application layers by D-PDU API imple- mentation** *[*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 implemen- tation, UDSonIP) application layers shall be supported if their respective application layer is available inside the ECU, all other application layers present in D-PDU API are optional.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01124]**{DRAFT} **Supported protocols by D-PDU API implementation** *[*ISO UDS on CAN with Application layer ISO\_15765\_3, ISO UDS on CAN with Ap- plication layer ISO\_14229\_3 (UDSonCAN) and ISO UDS on DoIP with Application layer ISO\_14229\_5 (UDSonIP) protocols shall be supported, all other protocols are optional.*♩(RS\_UCM\_00035)*

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

### 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.

**[SWS\_UCM\_01125]**{DRAFT} **Separation of D-PDU API-Software with the MVCI protocol module firmware** *[*A D-PDU API implementation may be split at OSI-Layer 4 into a D-PDU API implementation on OSI-Layer 5 (usually in the PC itself) and the VCI-Module on OSI-Layers 3 and 4 (usually the VCI itself).*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01126]**{DRAFT} **Root description file (RDF)** *[*Within an AUTOSAR Adaptive Platform, only one D-PDU API implementation is required for UCM, therefore the D-PDU API implementation may not use the D-PDU API root descrip- tion file (RDF).*♩(RS\_UCM\_00035)*

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

**[SWS\_UCM\_01127]**{DRAFT} **Module Description File (MDF)** *[*The D-PDU API im- plementation should not implement a protocol description file.*♩(RS\_UCM\_00035)*

The supported protocol module types are fixed in the UCM use case.

**[SWS\_UCM\_01128]**{DRAFT} **Symbolic names and IDs** *[*The Flashing Adapter may 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 may even hardcode (e.g. in a header file) all necessary IDs and operate the D-PDU API without symbolic names.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01129]**{DRAFT} **SAE J2534-1 and RP 1210a compatibility** *[*D-PDU API implementation may not be compatible to SAE J2534-1 and RP 1210a.*♩(RS\_- UCM\_00035)*

The Adaptive Platform does not need any migration path.

**[SWS\_UCM\_01130]**{DRAFT} **ComPrimitives in RawMode** *[*D-PDU API implemen- tation may not implement the IOCTL filter data structure.*♩(RS\_UCM\_00035)*

### Not required D-PDU API functions

PDULockResource() and PDUUnlockResource() are used to lock and unlock exclusive access to a ComLogicalLink in case of parallel usage of the D-PDU API implemen- tation 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.

**[SWS\_UCM\_01131]**{DRAFT} **PDUIoCtl(PDU\_IOCTL\_RESET)** *[*The parameter PDU\_IOCTL\_RESET may not be implemented in D-PDU API implementation so the call of PDUIoCtl(PDU\_IOCTL\_RESET) shall return the error code PDU\_ERR\_ID\_NOT\_SUPPORTED.*♩(RS\_UCM\_00035)*

### [SWS\_UCM\_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 the pa-

rameters PDU\_IOCTL\_START\_MSG, PDU\_IOCTL\_CLEAR\_MSG\_FILTER

and PDU\_IOCTL\_CLEAR\_MSG\_FILTER shall return the error code PDU\_ERR\_ID\_NOT\_SUPPORTED.*♩(RS\_UCM\_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.

### [SWS\_UCM\_01133]{DRAFT} PDUIoCtl(PDU\_IOCTL\_SEND\_BREAK)

*[*The IOCTL command PDU\_IOCTL\_SEND\_BREAK shall return PDU\_ERR\_ID\_NOT\_SUPPORTED.*♩(RS\_UCM\_00035)*

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.

**[SWS\_UCM\_01134]**{DRAFT} **Not used D-PDU API function return codes** *[*The return 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 may not be implemented into the

D-PDU API of the AUTOSAR Adaptive Platform.*♩(RS\_UCM\_00035)*

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 Adap- tive Platform, so PDU\_ERR\_API\_SW\_OUT\_OF\_DATE and PDU\_ERR\_MODULE\_FW\_OUT\_OF\_DATE return codes could not occur.

### Classic platform update with UCM Master and diagnostic tool



Vehicle

DoIP

Adaptive AUTOSAR Machine

Flashing Adapter

Diagnostic application

Classic

AUTOSAR Non

(ECU AUTOSAR

instance)

ISO 22900-2 (D-PDU API)

UCM

Master

Diagnostic Manager

VCI Module 1 VCI Module 2

CAN Ethernet

UDS on CAN

UDS on DoIP

Diagnostic Application is acting like an OTA Client

Communication medium

Classic AUTOSAR

Classic AUTOSAR

UCM

Vehicle State Manager

Driver Interface

Communication channel Ethernet or CAN/DoCAN Bus

Garage tester

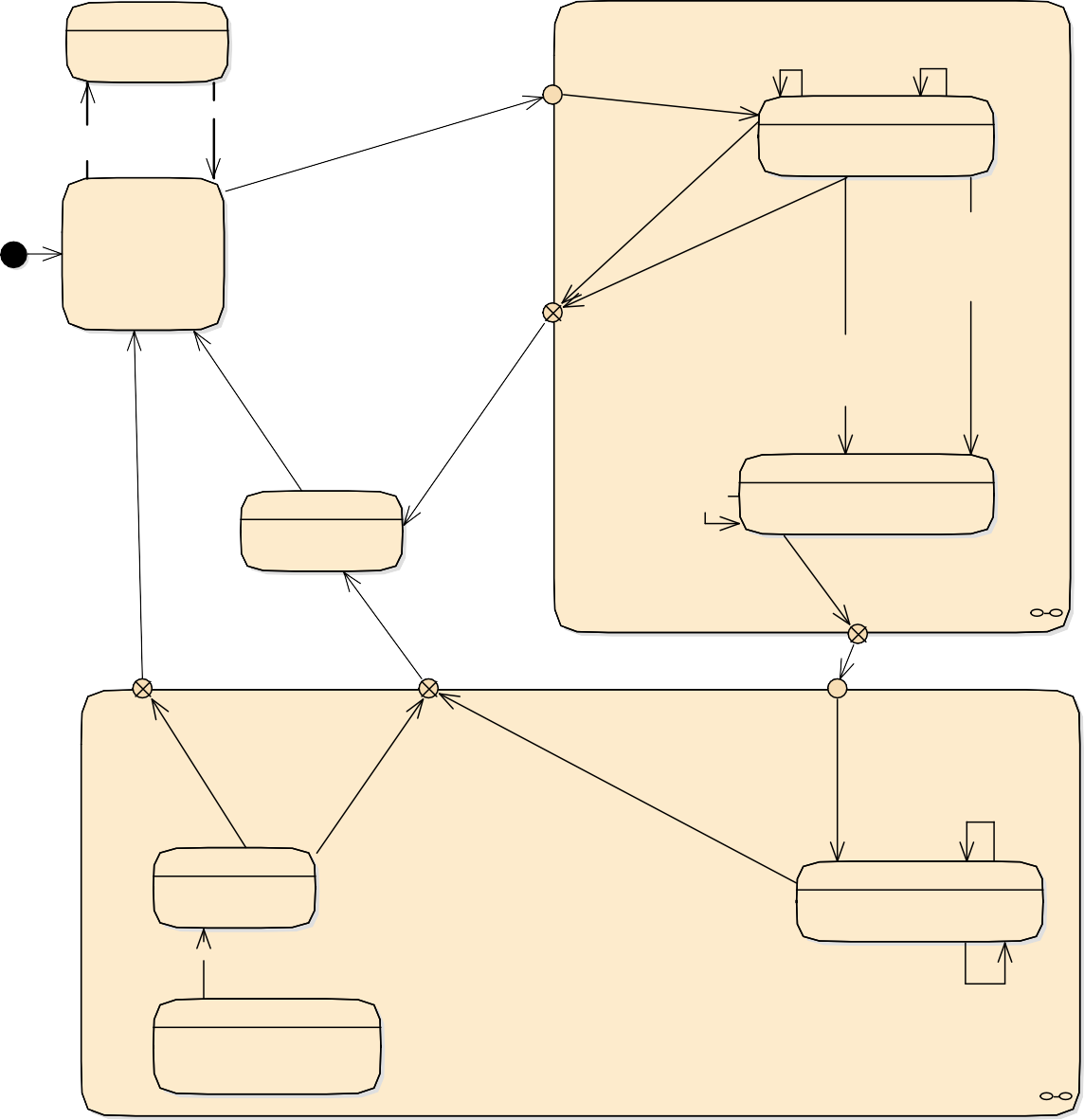
**Figure 7.7: Classic platform update with UCM Master and diagnostic tool**

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

Diagnostic Application is acting like an OTA Client

### Status reporting

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



**SYNCING**

**TRANSFERRING**

do / ComputeUpdates

Campaign

start

transferData()

transferExit() [Driver

Approval needed]

/ApprovalRequired = True

[SyncingDone]

**VEHICLEPACKAGE\_TRANSFERRING**

Sw Package Inventory(),

GetSw ClusterInfo()

do / VehiclePackageReceiving

transferVehicle Package()

**IDLE**

cancelCampaign(),

U CMMaster.delete Transfer()

Initi al

Campaign

aborted

transferExit()

[InvalidPackageManifest | Lack Resources | FailedDependency |

N onRecoverable Failure]

DriverApproval(True) [not

(InvalidPackageManifest | Lack Resources | FailedDependency)]

/ApprovalRequired = False

cancelCampaign()

transferExit() [not

(InvalidPackageManifest | Lack Resources | FailedDependency) & (Driver Approval not needed)]

[All CurrentStatus==Idle]

[InvalidPackage Manifest]

**CANCELLING**

All SWPs transferred

[Driver Approval needed]

/ApprovalRequired = True

**SOFTWAREPACKAGE\_TRANSFERRING**

do / Distribute SoftwarePackages

transferData(),

transferExit(), TransferStart

do / Cancelling

[(Start Proc. first SWP || A ll SWPs transferred)

& (Driver approval not needed)]

DriverApproval(True)

/ApprovalRequired = False

Campaign

successful

Transfer

finished

Campaign failed

**UPDATING**

U pdate

start

VehicleCheck(True) [All

CurrentStatus==Idle]

cancelCampaign (),Vehicle Check(False)

cancelCampaign(),

N onRecoverable Failure

All Packages processed successfully [All

Software Package States == kProcessed & Driver approval needed]

/ApprovalRequired = True

**VEHICLE\_CHECKING**

cancelCampaign(),

ActivationFailure

**PROCESSING**

do / VehicleSanityCheck

do / ProcessingSoftwarePackages

All CurrentStatus == Activated

All Packages processed successfully [All

Software Package States == kProcessed & Driver approval not needed]

**ACTIVATING**

do / UCM.Activate()

DriverApproval(True)

/ApprovalRequired = False

transferData(),

transferExit(), TransferStart

**Figure 7.8:** **Campaign State Machine (CampaignState field)**

Sw Package Inventory(), GetSw ClusterInfo()

**IDLE**

[SyncingDone]

transferVehicle Package()



**SYNCING**

do / ComputeUpdates

Campaign start

**TRANSFERRING**

Campaign aborted

Initi al

Transfer finished

[All CurrentStatus==Idle]

**CANCELLING**

do / Cancelling

Campaign successful

Campaign failed

U pdate start

**UPDATING**

**Figure 7.9:** **Campaign State Machine for OTA Client (TransferState field)**

Diagrams [7.8](#_bookmark123) and [7.9](#_bookmark124) do not include behaviour after reset ([[SWS\_UCM\_01205](#_bookmark107)] for more details)

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

**[SWS\_UCM\_01265] TransferState field** *[*UCM Master shall provide the state of a campaign over the TransferState field of the UCM Master’s VehiclePackageM- anagement service interface.*♩(RS\_UCM\_00042)*

**[SWS\_UCM\_01203] CampaignState field** *[*UCM Master shall provide the state of a campaign over the CampaignState field of the UCM Master VehicleDriver- ApplicationInterface and VehicleStateManagerInterface.*♩(RS\_UCM\_- 00042)* There is an overview of the campaign state machine in Fig. [7.8](#_bookmark123) detailing UCM Master campaign states and transitions.

### States

**[SWS\_UCM\_01204] Initial state** *[*UCM Master shall have kIdle default state.*♩(RS\_- UCM\_00035)*

**[SWS\_UCM\_01207] Trigger on kSoftwarePackage\_Transferring state** *[*On transition to kSoftwarePackage\_Transferring state and if all UCM subordinates part of the campaign are in kIdle state, UCM Master shall start or resume transfer- ring (TransferStart and TransferData as well as TransferExit if no streaming required) the software packages to the UCM subordinates according to the campaign orchestration.*♩(RS\_UCM\_00035, RS\_UCM\_00043)*

**[SWS\_UCM\_01209] Trigger on kProcessing state** *[*On transition to kProcessing state, UCM Master shall call ProcessSwPackage method to UCM subordinates to start or resume processing the software packages ready for processing according to the campaign orchestration.*♩(RS\_UCM\_00035, RS\_UCM\_00043)*

**[SWS\_UCM\_00210] Transferring of software packages on kProcessing state** *[*If UCM Master is in kProcessing state, UCM Master shall transfer Software Packages to the UCM subordinates according to the campaign orchestration.*♩(RS\_- UCM\_00035, RS\_UCM\_00043)*

**[SWS\_UCM\_01212] Trigger on kActivating state** *[*On transition to kActivat- ing state, UCM Master shall ask UCM subordinates to activate the software with Ac- tivate method call according to the campaign orchestration as configured in the VehiclePackage.*♩(RS\_UCM\_00035, RS\_UCM\_00043)*

**[SWS\_UCM\_01214]**{DRAFT} **Final action on kVehicleChecking state** *[*If UCM Master is in kVehicleChecking state and receives the method VehicleCheck call with parameter VehicleCheckResolution=True, UCM Master shall secondly commit (Finish) the software on all UCM subordinates part of the campaign.*♩(RS\_- UCM\_00035)*

**[SWS\_UCM\_01215]**{DRAFT} **Trigger on kCancelling state** *[*On transition to kCan- celling state, UCM Master shall rollback (Rollback) the activated or being verified Software Clusters, and revert the processed packages (RevertProcessedSw- Packages) of the UCM subordinates part of the campaign.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01216] Final action on kCancelling state** *[*If UCM Master is in kCancelling state and the rollback of software on all UCM subordinates is success- ful (successful Rollback and transition from kRollingBack to kRolledBack), UCM Master shall secondly commit (Finish) the software on all UCM subordinates part of the campaign.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01217] Monitoring of UCM subordinates** *[*UCM Master shall sub- scribe to the CurrentStatus field, in order to follow the current campaign from the state of the UCM Subordinates.*♩(RS\_UCM\_00035)*

### States Transitions

**[SWS\_UCM\_01218] Transition from kIdle state to kSyncing state** *[*If UCM Mas- ter is in kIdle state for CampaignState field and kIdle state for TransferState field, UCM Master shall enter the kSyncing state for CampaignState and kSync- ing for TransferState on a request to GetSwClusterInfo or SwPackageIn- ventory.*♩(RS\_UCM\_00035, RS\_UCM\_00033)*

**[SWS\_UCM\_01219] Transition from kSyncing state to kIdle state** *[*If UCM Mas- ter is in kSyncing state for CampaignState field and kSyncing state for Trans- ferState field, UCM Master shall enter the kIdle state and kIdle for Trans- ferState on completion of GetSwClusterInfo or SwPackageInventory.*♩(RS\_- UCM\_00035)*

**[SWS\_UCM\_01220] Transition from kIdle state to kVehiclePackageTransfer- ring and kTransferring states** *[*If UCM Master is in kIdle state for Campaign- State field and kIdle state for TransferState field, UCM Master shall enter the kVehiclePackageTransferring state and kTransferring state on successful completion of TransferVehiclePackage.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01221] Transition from kVehiclePackageTransferring state and kTransferring state to kCancelling state** *[*If UCM Master is in kVehi- clePackageTransferring state for CampaignState field and kTransferring state for TransferState field, UCM Master shall enter the kCancelling state for CampaignState and kCancelling state for TransferState on unsuccess- ful completion of TransferExit (Vehicle Package) or successful completion of DeleteTransfer (Vehicle Package) or non recoverable error of Transfer- Data.*♩(RS\_UCM\_00035, RS\_UCM\_00039)*

**[SWS\_UCM\_01222] Transition from kVehiclePackageTransferring state to kSoftwarePackage\_Transferring state** *[*If UCM Master is in kVehiclePack- ageTransferring state, UCM Master shall enter the kSoftwarePackage\_- Transferring state on successful completion of TransferExit (Vehicle Pack- age).*♩(RS\_UCM\_00035, RS\_UCM\_00037, RS\_UCM\_00038)*

**[SWS\_UCM\_01227] Transition from kSoftwarePackage\_Transferring state and kTransferring state to kCancelling state** *[*If UCM Master is in kSoft- warePackage\_Transferring state for CampaignState and kTransferring state for TransferState, UCM Master shall enter the kCancelling state for Cam- paignState and kCancelling state for TransferState on successful cancella- tion request (CancelCampaign) or if there is a non recoverable transfer failure from one of the UCM subordinates.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01228] Transition from kSoftwarePackage\_Transferring state and kTransferring state to kProcessing state and kUpdating state** *[*When UCM Master 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 UCM subordinates are at state kTransferred or at least one Software Package started being processed by

ProcessSwPackage call to one UCM subordinate according to the campaign orches- tration, UCM Master shall enter the kProcessing state for CampaignState and kUpdating state for TransferState.*♩(RS\_UCM\_00035, RS\_UCM\_00037, RS\_- UCM\_00038, RS\_UCM\_00043)*

**[SWS\_UCM\_01229]**{DRAFT} **SafetyConditions while processing stream** *[*In the case there is transition from kSoftwarePackage\_Transferring state to kPro- cessing state, the SafetyConditions for kProcessing state shall apply even though there are Software Packages transferring.*♩(RS\_UCM\_00035, RS\_UCM\_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\_UCM\_01234] Transition from kProcessing state to kActivating state** *[*If UCM Master is in kProcessing state and all software packages of the campaign have been successfully (successful ProcessSwPackage) processed and all UCM subordinates part to the campaign are in the kReady state, UCM Master shall en- ter the kActivating state.*♩(RS\_UCM\_00035, RS\_UCM\_00037, RS\_UCM\_00038)*

**[SWS\_UCM\_01236] Transition from kProcessing state and kUpdating state to kCancelling state** *[*If UCM Master is in kProcessing state for Campaign- State and kUpdating state for TransferState, UCM Master shall enter the kCancelling state for CampaignState and kCancelling state for Transfer- State on successful cancellation request (CancelCampaign) or in case of non re- coverable processing failure of one of the UCM subordinates.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01239] Transition from kActivating state and kUpdating state to kCancelling state** *[*If UCM Master is in kActivating state for Campaign- State and kUpdating state for TransferState, UCM Master shall enter the kCancelling state for CampaignState and kCancelling state for Transfer- State if any UCM subordinates part of the campaign unsuccessfully (unsuccessful Activate and transition from kVerifying to kRollingBack) completed activa- tion.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01240] Transition from kActivating state to kVehicleChecking state** *[*If UCM Master is in kActivating state, UCM Master shall enter the kVehi- cleChecking state if all UCM subordinates part of the campaign successfully (suc- cessful Activate and transition from kVerifying to kActivated) completed acti- vation.*♩(RS\_UCM\_00035, RS\_UCM\_00037)*

**[SWS\_UCM\_01241]**{DRAFT} **Transition from kVehicleChecking state and kUp- dating state to kCancelling state** *[*If UCM Master is in kVehicleChecking state for CampaignState and kUpdating state for TransferState, and receives the method VehicleCheck call with parameter VehicleCheckResolution=False, UCM Master shall enter the kCancelling state for CampaignState and kCan- celling state for TransferState.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01242] Transition from kVehicleChecking state and kUpdating state to kIdle state** *[*If UCM Master is in kVehicleChecking state for Campaign- State and kUpdating state for TransferState and all UCM subordinates part of the campaign transitioned from kCleaningUp to kIdle, UCM Master shall enter the kIdle state for CampaignState and kIdle state for TransferState.*♩(RS\_UCM\_- 00035)*

**[SWS\_UCM\_01243] Transition from kCancelling state to kIdle state** *[*If UCM Master is in kCancelling state for CampaignState and kCancelling state for TransferState and all UCM subordinates part of the campaign transitioned from kCleaningUp to kIdle, UCM Master shall enter the kIdle state for Campaign- State and kIdle state for TransferState.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01246]**{DRAFT} **Unreachable UCM during update campaign** *[*In case a UCM is not reachable by UCM Master during an update campaign (from kTrans- ferring or kUpdating), UCM Master shall transit to kCancelling state for Cam- paignState and kCancelling state for TransferState.*♩(RS\_UCM\_00035, RS\_- UCM\_00037)*

### Campaign cancelling

CancelCampaign method could be used at garage to unlock a blocked update. De- tails on action by UCM Master, 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. UCM Master could list transferred Software Packages by calling the UCM subordinates with GetSwPackages.

**[SWS\_UCM\_01244]**{DRAFT} **Cancellation of an update campaign shall be pos- sible** *[*Method CancelCampaign from UCM Master shall trigger a campaign can- cel from kTransferring or kUpdating states (from TransferState field).*♩(RS\_- UCM\_00035, RS\_UCM\_00037)*

**[SWS\_UCM\_01270]**{DRAFT} **New campaign disabling** *[*UCM Master shall remain in kIdle when a CancelCampaign method has been called with DisableCampaign parameter set.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01271]**{DRAFT} **New campaign enabling** *[*Method AllowCampaign from UCM Master shall reallow new campaign after a CancelCampaign method was called with DisableCampaign parameter set.*♩(RS\_UCM\_00035)*

**[SWS\_UCM\_01273]**{DRAFT} **CancelCampaign CancelFailed error** *[*Cancel- Campaign shall raise the error ApplicationError CancelFailed in case can- celling of a campaign fails.*♩(RS\_UCM\_00020)*

**[SWS\_UCM\_01274]**{DRAFT} **CancelCampaign OperationNotPermitted error** *[*CancelCampaign shall raise the error ApplicationError OperationNotPer- mitted in case the UCM Master states are at kIdle, kSyncing or kCancelling.*♩ (RS\_UCM\_00020)*

### Campaign Reporting

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

**[SWS\_UCM\_01247] Method to read History Report** *[*UCM Master shall provide a method GetCampaignHistory to retrieve all actions that have been performed by UCM Master when exiting state kUpdating from a specific time window.*♩(RS\_UCM\_- 00034)*

**[SWS\_UCM\_01248] Content of History Report** *[*UCM Master 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 and the campaign result in CampaignHistoryType*♩(RS\_UCM\_00034)*

The elements of HistoryVectorType [SWS\_UCM\_00254] are not sorted.

**[SWS\_UCM\_01266]**{DRAFT} **Subordinate Not Available On The Network** *[* UCM Master shall record persistently the error kSubordinateNotAvailableOn- TheNetwork in case one of the UCM subordinate involved in the current campaign stops offering its service interface and later report it with GetCampaignHistory.*♩ (RS\_UCM\_00034)*

**[SWS\_UCM\_01267]**{DRAFT} **Vehicle State Manager Communication Error** *[*UCM Master shall record persistently the error kVehicleStateManagerCommunica- tionError in case the communication with Vehicle State Manager is not possible and later report it with GetCampaignHistory.*♩(RS\_UCM\_00034)*

**[SWS\_UCM\_01268]**{DRAFT} **Vehicle Driver Interface Communication Error** *[*UCM Master shall record persistently the error kVehicleDriverInterfaceCommuni- cationError in case the communication with Vehicle Driver Interface is no longer possible and later report it with GetCampaignHistory.*♩(RS\_UCM\_00034)*

**[SWS\_UCM\_01269]**{DRAFT} **Campaign cancellation history** *[*If CancelCampaign method is called, UCM Master shall record persistently this event to later report it with GetCampaignHistory.*♩(RS\_UCM\_00034)*

### Content of Vehicle Package

|  |
| --- |
| Executables |
| Data |
| Manifests |

|  |
| --- |
| Executables |
| Data |
| Manifests |

**Figure 7.10: Vehicle package overview**

Software Package A Signed container

SoftwareCluster A

Signed container

Software Package B Signed container

SoftwareCluster B

Signed container

Software Cluster Manifest

Authentication tag

Vehicle Package

Signed ARXML file Software Package

**manifest** A

Software Package

**manifest** B

Software Cluster Manifest

Authentication tag

Software Package Manifest

Authentication tag

Vehicle Package manifest

OEM authentication tag

Software Package Manifest

Authentication tag

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 Pack- age Manifests extracted from Backend packages stored in the Backend database. These Software Packages have to be modelled as a so-called SoftwarePack- age 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 UCM Master 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 Pack- age 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 UCM Master 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 UCM Mas- ter 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, UCM Master will be informed at any time of campaign if vehicle safety is met or not.
* UCM Master identifiers list: defines backup UCM Masters
* Campaign orchestration: You can refer to [[10](#_bookmark10)] for more details. This campaign model allows to group activation of several UCMs and group Software Pack- ages processing and transferring.

**[SWS\_UCM\_01301]**{DRAFT} **Vehicle Package authentication** *[*The Vehicle Package shall be successfully authenticated by the UCM Master using CryptoSer- viceCertificate at TransferData or TransferExit call before any transfer of the Software Packages, otherwise raise ApplicationError OperationNot- Permitted.*♩(RS\_UCM\_00039, RS\_UCM\_00043)*

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

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

**[SWS\_UCM\_01303]**{DRAFT} **Dependencies between Software Packages** *[*UCM Master shall check dependencies based on Vehicle Package Manifests and Software Packages Manifests before an transfer of Software Packages.*♩ (RS\_UCM\_00035, RS\_UCM\_00043)*

In case UCM Master does not handle any recovery from a failing dependency check, UCM Master could cancel the active campaign and return ApplicationError MissingDependencies from TransferExit. UCM Master could avoid returning MissingDependencies error from TransferExit by resolving missing dependen- cies. For instance, UCM Master could request to install missing Software Cluster using RequestedPackage field, or remove conflicting Software Cluster.

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

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

**[SWS\_UCM\_01307]**{DRAFT} **Vehicle Package format not supported** *[*UCM Mas- ter shall return ApplicationError UnsupportedPackageFormat from Trans- ferData or TransferExit methods call in the case the Vehicle Package format is not supported.*♩(RS\_UCM\_00043)*

**[SWS\_UCM\_01306] TransferExit Invalid package manifest** *[*TransferExit shall raise the error ApplicationError InvalidPackageManifest upon recep- tion of an invalid manifest.*♩(RS\_UCM\_00012)*

**[SWS\_UCM\_01308]**{DRAFT} **Check Vehicle Package version compatibility against UCM Master version** *[*At TransferExit call, UCM Master shall raise Applica- tionError IncompatiblePackageVersion if the version for the Vehicle Pack- age transferred expressed by minimumSupportedUcmMasterVersion attribute is higher than the current version of UCM Master expressed by version attribute.*♩(RS\_- UCM\_00043)*

### Vehicle update security and confidentiality

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

**[SWS\_UCM\_CONSTR\_00013]**{DRAFT} **Confidential information protection** *[*The VehiclePackageManagement and VehicleDriverApplicationInterface in- terfaces shall only be called over secure communication channel providing confiden- tiality protection.*♩(RS\_UCM\_00033)*

The GetSwClusterInfo, SwPackageInventory, GetCampaignHistory, GetSwClusterChangeInfo, GetHistory, GetSwClusterDescription and GetSwPackages methods are using data that could identify vehicle user and there- fore should be protected for confidentiality.

# API specification

There are no APIs defined in this release.

# Service Interfaces

|  |
| --- |
| « enumeration»  **Sw Package StateType** |
| k Transferring = 0x00 k Transfered = 0x01 k Processing = 0x02 k Processed = 0x03 k ProcessingStream = 0x04 |
| *requirements*  SWS\_U CM\_00038 |

## Type definitions

This chapter lists all types provided by the UCM.

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



PackageManagement

« service»

**UCM**

SWS\_U CM\_00131

*requirements*

+ GetSw ClusterInfo(): Sw ClusterInfoVectorType

+ GetSw ClustersChangeInfo(): Sw ClusterInfoVectorType

+ GetSw ClusterDescription(): Sw DescVectorType

+ GetSw Packages(): Sw PackageInfoVectorType

+ TransferStart(uint64, TransferIdType\*, unsigned int\*): void

+ TransferData(TransferIdType, ByteVectorType, uint32): void

+ TransferExit(TransferIdType): void

+ DeleteTransfer(TransferIdType): void

+ ProcessSw Package(TransferIdType): void

+ GetSw ProcessProgress(TransferIdType): char

+ RevertProcessedSw Packages(): void

+ Cancel(TransferIdType): void

+ Activate(): void

+ Rollback(): void

+ Finish(): void

+ GetHistory(unsigned long, unsigned long): GetHistoryVectorType

+ GetId(): U CMIdentifierType

+ CurrentStatus: PackageManagerStatusType

«interface»

**PackageManagement**



*requirements*

SWS\_U CM\_00077

k Present = 0x00 kAdded = 0x01 k Updated = 0x02 kRemoved = 0x03

« enumeration»

**Sw ClusterStateType**

*requirements*

SWS\_U CM\_00078

+ name: Sw N ameType

+ version: StrongRevisionLabelString

+ state: Sw ClusterStateType

« dataType»

**Sw ClusterInfoType**

*requirements*

SWS\_U CM\_00031

«primitive»

**TransferIdType**

*requirements*

SWS\_U CM\_00175

« dataType»

**StrongRevisionLabelString**

*requirements*

SWS\_U CM\_00173

«primitive»

**UCMIdentifierType**

*requirements*

SWS\_U CM\_00032

«primitive»

**Byte VectorType**

*requirements*

SWS\_U CM\_00071

« dataType»

**Sw NameType**

|  |
| --- |
| « enumeration»  **Package ManagerStatusType** |
| kIdle = 0x00 k Ready = 0x01 k Processing = 0x02 k Activating = 0x03 k Activated = 0x04  k RollingBack = 0x05 k RolledBack = 0x06 k CleaningUp = 0x07 kVerifying = 0x08 |
| *requirements*  SWS\_U CM\_00044 |

|  |
| --- |
| « dataType»  **Sw ClusterInfoVectorType** |
| + sw InfoTypes: Sw ClusterInfoType [1..n] |
| *requirements*  SWS\_U CM\_00079 |

|  |
| --- |
| « dataType»  **GetHistoryType** |
| + time: uint64  + name: Sw N ameType  + version: StrongRevisionLabelString  + action: ActionType  + resolution: ResultType  + error: unsigned short |
| *requirements*  SWS\_U CM\_00134 |

|  |
| --- |
| « dataType»  **GetHistoryVectorType** |
| + history: GetHistoryType [1..n] |
| *requirements*  SWS\_U CM\_00135 |

**Figure 9.1: UCM Subordinate composite structure**



*requirements*

SWS\_U CM\_00186

*requirements*

SWS\_U CM\_00187

+ sw DescTypes: Sw DescType [1..n]

« dataType»

**Sw DescVectorType**

+ name: Sw N ameType

+ version: StrongRevisionLabelString

+ typeApproval: string

+ license: string

+ releaseN otes: string

+ size: unsigned long

« dataType»

**Sw DescType**

*requirements*

SWS\_U CM\_00040

+ sw PackageInfo: Sw PackageInfoType [1..n]

« dataType»

**Sw Package InfoVectorType**

*requirements*

SWS\_U CM\_00039

+ name: Sw N ameType

+ version: StrongRevisionLabelString

+ transferId: TransferIdType

+ consecutiveBytesReceived: unsigned long

+ consecutiveBlocksReceived: unsigned long

+ state: Sw PackageStateType

« dataType»

**Sw PackageInfoType**



|  |
| --- |
| « enumeration»  **ActionType** |
| k Update = 0x00 k Install = 0x01 kRemove = 0x02 |
| *requirements*  SWS\_U CM\_00132 |

|  |
| --- |
| « enumeration»  **ResultType** |
| k Activated = 0x00 k ActivatedAndRolledBack = 0x01 k VerificationFailed = 0x02 |
| *requirements*  SWS\_U CM\_00133 |

*(from UCM API)*

kIdle = 0x00

k VehiclePackageTransferring = 0x01 k SoftwarePackageTransferring = 0x02 kUpdating = 0x03

k Cancelling = 0x04

« use»

*requirements*

SWS\_U CM\_01178

**Vehicle State**

**Manager**

« use»

*(from UCM API)*

« enumeration»

**TransferStateType**

«interface»

V ehiclePackageManagement

« enumeration»

**SafetyStatesType**

*requirements*

SWS\_U CM\_01114

«primitive»

**SafetyConditionType**

*(from UCM API)*

**OTA Client**

Safe = 0x00

N otSafe = 0x01

**Vehicle::ECU::Machine::AUTOSAR Adaptive Platform::UCM Master::VSM::VehicleStateManager**

+ Safety Conditions: Safety ConditionsVectorType

+ CampaignState: CampaignStateType

+ PublishSafety State(Safety StatesVectorType): void

+ VehicleCheck(bool): void

*requirements*

N otSupported = 0x02

*requirements*

SWS\_U CM\_01138

« dataType»

**SafetyConditionsVectorType**

+ safety Conditions: Safety ConditionsType [1..n]

« dataType»

**CampaignHistoryVectorType**

+ campaignHistory: CampaignHistoryType [1..n]

*requirements*

SWS\_U CM\_00296

SWS\_U CM\_00183

VehicleStateManager

SWS\_U CM\_01136

*requirements*

VSM

*requirements*

SWS\_U CM\_01177

kIdle = 0x00 kSyncing = 0x01

k VehiclePackageTransferrring = 0x02 k SoftwarePackageTransferring = 0x03 k Processing = 0x04

k Activating = 0x05 k VehicleChecking = 0x06 k Cancelling = 0x07

« enumeration»

**CampaignStateType**

VPM

« service»

**UCM Master**

VDI

« dataType»

**SafetyStatesVectorType**

+ safety States: Safety StatesType [1..n]

*requirements*

SWS\_U CM\_01137

**Vehicle Driver Interface**

« dataType»

ara:: com

« use»

VehicleDriverApplication

« dataType»

**CampaignHistoryType**

**:**

pe

|  |
| --- |
| «interface»  **Vehicle::ECU::Machine::Autosar Adaptive Application layer::Vehicle Driver Interface**  **:ara::com::VehicleDriverApplication** |
| + ApprovalRequired: bool  + Safety State: Safety StatesVectorType  + Safety Conditions: Safety ConditionsVectorType  + CampaignState: CampaignStateType |
| + DriverApproval(bool, SafetyConditionType)  + GetSw PackageDescription(): Sw PackageDescVectorType  + GetSw ClusterDescription(): Sw DescVectorType  + CancelCampaign(bool): void  + Allow Campaign(): void  + GetCampaignHistory(unsigned long, unsigned long): CampaignHistoryVectorTy  + GetSw TransferProgress(): unsigned int  + GetSw ProcessProgress(): unsigned int |
| *requirements*  SWS\_U CM\_00182 |

+ campaignResult: CampaignResultType

+ history Vector: HistoryVectorType

*requirements*

SWS\_U CM\_00251

« dataType»

**HistoryVectorType**

**Sw DescVectorType**

+ sw DescTypes: Sw DescType [1..n]

*requirements*

SWS\_U CM\_00187

*(from UCM API)*

« dataType»

**Sw Package DescVectorType**

+ sw PackageDescription: Sw PackageDescType [1..n]

*requirements*

SWS\_U CM\_00269

« dataType»

**GetHistoryVectorType**

+ history: GetHistoryType [1..n]

*requirements*

SWS\_U CM\_00135

*(from UCM API)*

« dataType»

**HistoryType**

+ U CMid: UCMIdentifierType

+ U CMHistory: GetHistoryVectorType

*requirements*

SWS\_U CM\_00290

+ history: HistoryType [1..n]

*requirements*

SWS\_U CM\_00254

« dataType»

**CampaignResultType**

+ campaignResolution: U cm MasterResolutionType

+ U CMStepError: U CMStepErrorType

+ campaignResolutionTime: unsigned long

+ campaignStartTime: unsigned long

*requirements*

SWS\_U CM\_00252

« dataType»

**Sw PackageDescType**

+ sw Desc: Sw DescType

+ packageAction: ActionType

+ duration: int

*requirements*

SWS\_U CM\_00268

« enumeration»

**UCM API::ResultType**

k Activated = 0x00 k ActivatedAndRolledBack = 0x01 k VerificationFailed = 0x02

*requirements*

SWS\_U CM\_00133

SWS\_U CM\_00253

« enumeration»

**UCM API::ActionType**

k Update = 0x00 k Install = 0x01 kRemove = 0x02

*requirements*

SWS\_U CM\_00132

« dataType»

**GetHistoryType**

+ time: uint64

+ name: Sw N ameType

+ version: StrongRevisionLabelString

+ action: ActionType

+ resolution: ResultType

+ error: unsigned short

*requirements*

SWS\_U CM\_00134

*(from UCM API)*

« enumeration»

**Ucm MasterResolutionType**

k Success = 0x00 kUCMError = 0x01

*requirements*

SWS\_U CM\_00291

+ U CMStepError: U CMStepErrorType

« dataType»

**UCMStepErrorVectorType**

k InvalidVehiclePackageManifest = 0x02 k SubordinateN otAvailableOnTheNetwork = 0x03 k VehicleStateManagerCommunicationError = 0x04 k VehicleDriverInterfaceCommunicationError = 0x05 k CampaignCancelled = 0x06

*requirements*

SWS\_U CM\_00256

« enumeration»

**Software PackageStepType**

« dataType»

**UCMStepErrorType**

k Transfer = 0x00 k Process = 0x01 k Activate = 0x02

+ U CMid: UCMIdentifierType

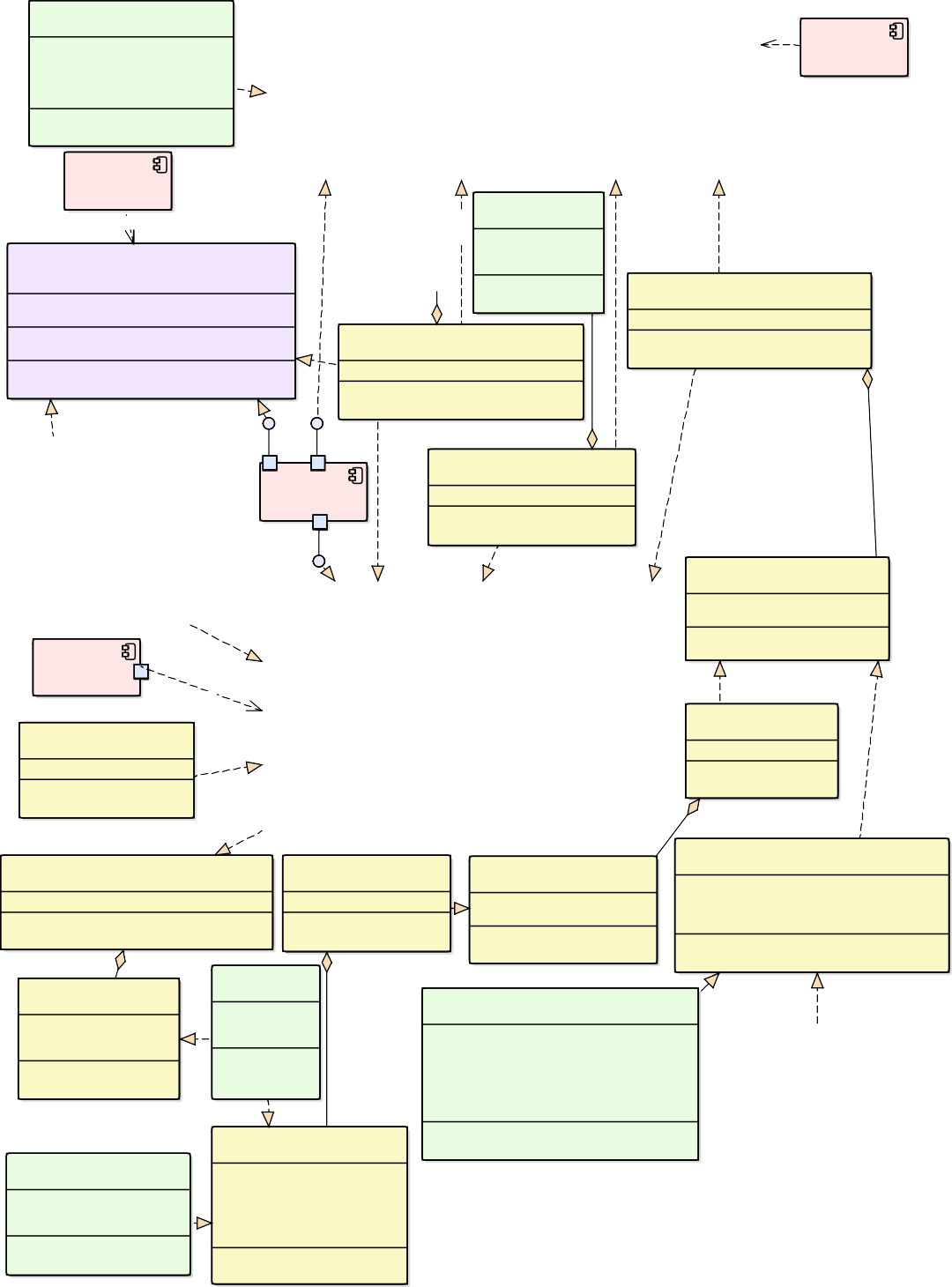
+ softwarePackageStep: SoftwarePackageStepType

+ returnedError: unsigned int

*requirements*

SWS\_U CM\_00255

*requirements*

**Figure 9.2: UCM Master composite structure**



*requirements*

SWS\_U CM\_00031

«primitive»

**TransferIdType**

*requirements*

SWS\_U CM\_00079

+ sw InfoTypes: Sw ClusterInfoType [1..n

« dataType»

**Sw ClusterInfoVectorType**

SWS\_U CM\_00181

*requirements*

+ Sw PackageInventory(Sw N ameVersionVectorType): Sw N ameVersionVectorType

+ GetSw ClusterInfo(): Sw ClusterInfoVectorType

+ GetSw Packages(): Sw PackageInfoVectorType

+ TransferStart(Sw N ameType, TransferIdType\*, unsigned int\*): void

+ TransferVehiclePackage(unsigned long, TransferIdType\*, unsigned int\*): void

+ TransferData(TransferIdType, ByteVectorType, unsigned long): void

+ TransferExit(TransferIdType): void

+ DeleteTransfer(TransferIdType): void

+ Allow Campaign(): void

+ CancelCampaign(bool): void

+ GetCampaignHistory(unsigned long, unsigned long): CampaignHistoryVectorType

+ TransferState: TransferStateType

+ RequestedPackage: Sw N ameVersionType

+ Safety State: Safety StatesVectorType

+ Safety Conditions: Safety ConditionsVectorType

«interface»

**Vehicle::ECU::Machine::AUTOSAR Adaptive Platform::UCM Master::VPM::VehiclePackageManagement**

*requirements*

SWS\_U CM\_00176

SWS\_U CM\_00177

*requirements*

« dataType»

**Sw NameVersionType**

« dataType»

**Sw Name VersionVectorType**

*requirements*

SWS\_U CM\_00040

+ sw PackageInfo: Sw PackageInfoType [1..n

« dataType»

**Sw Package InfoVectorType**

]

+ sw N ameVersionVectorType: SwNameVersionType [1..n]

+ name: Sw N ameType

+ version: StrongRevisionLabelString

]

# Interfaces to other Functional Clusters (informative)

## 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. In addi- tion, the standardized public interfaces which are accessible by user space applications (see chapter [8](#_bookmark131)) can also be used for interaction between Functional Clusters.

The goal is to provide a clear understanding of Functional Cluster boundaries and in- teraction, without specifying syntactical details. This ensures compatibility between documents specifying different Functional Clusters and supports parallel implementa- tion of different Functional Clusters. Details of the interfaces are up to the platform provider.

## Interfaces Tables

### UCM update notification

UCM shall provide the notification to other Functional Clusters that changes have been done to the software. This enables other functional clusters to check if updated man- ifests have changes relevant for the concerned Functional Cluster. This can be done through the field CurrentStatus provided by the UCM service.

# Packages distribution within vehicle detailed sequence examples

## Collect information of present Software Clusters in vehicle

From a regular basis, UCM master and UCM can collect information of present Soft- ware 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.



**sd** [U seCase] 1. Determine installed SWCL in vehicle [Determine installed SWCL in vehicle]

Diagnostic tool

*(from Actors)*

Adaptive platform A

U CM master

*(from Actors)*

Adaptive platform B

UCM slave 1

*(from Actors)*

GetSw ClusterInfo()

:Sw ClusterInfoVector

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

## 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 communi- cation is established, for instance for security purpose.

Communication trial between Backend and UCM master can be done on driver’s re- quest or from a scheduler.

### Pull package from Backend into vehicle

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

OTA Client

U CM master

UCM sub 1

UCM sub 2



*(from Actors)*

*(from Actors)*

*(from Actors)*

*(from Actors)*

OTA Client could regularly (scheduler

or diag trigger) establish connection with backend and update whole vehicle installed SWCluters

Adaptive Platform A Adaptive Platform B

Adaptive Platform C

Once connection is set with OTA Client,

Backend is sending its inventory based on VIN already communicated by OTA client

:TransferState=IDLE

UCM Master computes what

Compute U pdates() SWCLs should be updated

Backend sends

Vehicle Package

transferExit(transferId)

Compute Dependencies()

:transferExitReturn

transferData(transferId, block, blockCounter)

:transferDataReturn

**loop**

:Sw N ame VersionVector

:TransferState=IDLE TransferVehicle Package(Size)

:TransferState=Transferring

:transferId

Sw Package Inventory(Sw N ame VersionVector)

Merge Sw ClusterInfoVectors()

rInfoVector

:Sw Cluste

GetSw Cl

:Sw ClusterInfoVector

usterInfo()

GetSw ClusterInfo()

**onal**

**opt Opti**

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

### Push package from backend into vehicle

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

OTA Client

U CM master

UCM sub 1

UCM sub 2



*(from Actors)*

*(from Actors)*

*(from Actors)*

*(from Actors)*

:TransferState=IDLE

GetSw ClusterInfo()

Backend is requesting installed

SWClusters in vehicle

**opt Optionnal**

GetSw ClusterInfo()

:Sw ClusterInfoVector

GetSw ClusterInfo()

:Sw ClusterInfoVector

Merge Sw ClusterInfoVectors()

:Sw ClusterInfoVector

BackendCompute U pdatesAndDependencies() transferVehicle Package(Size)

:TransferState=Transferring

:transferId

Backend sends

Vehicle Package

**loop**

transferData(transferId, block, blockCounter)

:transferDataReturn

transferExit(transferId)

:transferExitReturn

Compute Dependencies()

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

## Packages transfer from backend into targeted UCM

Adpative platform B

Adaptive platform C

Adaptive Platform A

OTA Client Driver Interface

*(from Actors) (from Actors)*

U CM master Vehicle State Manager

*(from Actors) (from Actors)*

UCM sub 1

*(from Actors)*

UCM sub 2

*(from Actors)*

:TransferState = IDLE

TransferVehicle Package(Size)

:CampaignState=Vehicle Package\_Transferring

:TransferState = Transferring

:transferId

**loop transfer blocks**

transferData(transferId, block, blockCounter)

:transferDataReturn

transferExit(transferId)

:transferExitReturn

Temporary Store Vehicle PackageManifest()

ManifestAuthentication(Signature) Parse SWCLPackage Manisfests(): ((UCM Slave1 Id,

SWCLPackageBSize),( UCM slave2 Id, SWCLPackageBSize))

:CampaignState = VehiclePackage\_Transferring

**opt Vehicle Safety condition**

subcribe(Safety Conditions)

Safety State(Safety StatesVector)

**opt Driver notification**

:ApprovalRequired==True

:CampaignState = VehiclePackage\_Transferring

subscribe(Safety Conditions) GetSw ClusterDescription()

GetSw ClusterDescription()

GetSw ClusterDescription()

:Sw DescVectorType

:Sw DescVectorType

:Sw DescVectorType

GetSw Package Description()

Get Metadata from Vehicle Package Manifest()

:Sw DescVectorType

WaitApproval()

DriverApproval(Approval, SafetyStates)

:CampaignState = SoftwarePackage\_Transferring

**par Transfers**

:RequestedPackage == SWCLPackageAName

:SafetyState=True

TransferStart(SWCLPackage AN ame)

Counter argument for

efficient resume

TransferStart(SWCLPackageASize)

check AvailableMemory()

:transferId1

:transferId

**loop Streaming of package A**

transferData(transferId1, block, BlockCounter)

If upgrade, Check

InstalledVersion < NewVersion

transferData(transferId1, block, blockCounter)

Package Authentication(PackageSignature)

:transferDataReturn

:transferDataReturn

transferExit(transferId1)

transferExit(transferId1)

Consistency Check(Checksum)

:transferExitReturn

:TransferExitReturn

**opt Progress check**

GetSw TransferProgress()

:Progress

:RequestedPackage == SWCLPackageBName

TransferStart(SWCLPackageBName)

TransferStart(SWCLPackageBSize)

CheckAvailable Memory()

:transferId2

:transferId2

**loop Streaming of package B**

transferData(transferId2, Block, BlockCounter)

transferData(transferId2, Block, BlockCounter)

:transferDataReturn

:transferDataReturn

transferExit(transferId2)

transferExit(TransferId2)

Package Authentication

(Package Signature)

transferExitReturn()

:transferExitReturn

Check Version

(Version, PreviousVersion)

:TransferState=Updating

:CampaignState=Processing

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

## Package processing

Driver Interface



*(from Actors)*

U CM master

*(from Actors)*

Vehicle State Manager

*(from Actors)*

UCM sub 1

*(from Actors)*

UCM sub 2

*(from Actors)*

:CampaignState = SoftwarePackage\_Transferring

Subscribe(CurrentStatus)

:CurrentStatus=ready

Parse Vehicle Package Manifest(): CampaignOrchestration, Dependencies

CheckVehicle SWCLDependencies (Vehicle Package Dependencies)

**opt Vehicle driver notification for processing**

:CampaignState = SoftwarePackage\_Transferring

:ApprovalRequired==True

WaitApproval()

DriverApproval(True, SafetyStates)

GetSw ProcessProgress()

:Progress

**opt Vehicle safety state**

subscribe(Safety Conditions)

Safety State(Safety StatesVector)

:Safety StatesVector

Wait for Safe conditions()

Safety State(Safety StatesVector)

:Safety StatesVector

:CampaignState = Processing

**par Processes packages**

**opt Check on-going processing**

GetSw Packages(): Sw ClusterInfoVectorType

ProcessSw Package(transferId1): ProcessSwPackageReturnType

Package IntegrityCheck()

Parse ActionFrom Manifest()

ManifestConsistency Check()

:ProcessSw Package Return

CheckSWCLAvailableMemory (SWCLPayloadSize)

**loop Until Progress = 100**

GetProcessProgress(transferId1): ProcessingStatusType

:progress=100

**opt Check on-going processing**

GetSw Packages(Sw InfoN ame2): SwClusterInfoVectorType

ProcessSw Package(transferId2): ProcessSwPackageReturnType

Package IntegrityCheck()

CheckSWCLAvailableMemory (SWCLPayloadSize)

ManifestConsistency Check()

Parse ActionFrom Manifest()

**loop Until Progress = 100**

GetProcessProgress(transferId2): ProcessingStatusType

:progress=100

:CampaignState = Activating

**Figure C.5: Packages processing by UCMs**

## Package activation

Driver Interface

U CM master

Vehicle State Manager

UCM sub 1

UCM sub 2



*(from Actors)*

*(from Actors)*

*(from Actors)*

*(from Actors)*

*(from Actors)*

:CampaignState=Processing

ssing

WaitApproval()

es)

**par Activatio**

**l**

:CampaignState=kActivating

tStatus)

:READY

Partition activation

Activate(ActivationMethod)

:CurrentStatus = k

Define ActivationMethod(Manifest or ActionMethod)

CheckPackage Dependencies(SWCLDependencies)

:0

:CurrentStatus = k

:Vehicle Sta

swapPartitionAndSyncThem()

:CurrentStatus = k

Sw restart activation

Subscribe(CurrentStatus)

Method)

:CurrentStatus = kActivating

Define ActivationMethod

(Manifest or ActionMethod)

CheckPackage Dependencies (SWCLDependencies)

:CurrentStatus = kVerifying

stopOldSWCLIfNeeded()

startN ew SWCLIfNeeded()

:CurrentStatus = kActivated

:CampaignState=Vehicle\_Chec

leaningUp

kIdle

Finish()

:CurrentStatus = kCleaningUp

:CurrentStatus = kIdle

:CampaignState=kIdle

:CurrentStatus =

:CurrentStatus = kC

Finish()

Vehicle Checks()

king

:0

Activate(Activation

:READY

Activated

te

Verifying

Activating

Subscribe(Curren

**ns could be performed in paralle**

**/during activation**

:Safe ToUpdate=True

**Vehicle Safety conditions before**

subscribe(Safety Conditions)

**opt Check**

riverApproval(True, SafetyStat

ApprovalRequired=True()

**onal Action**

:CampaignState=k Proce

D

**opt Opti**

**Figure C.6: Packages activation by UCMs**

## Package rollback

OTA Client

Vehicle State ManagerDriver Interface

U CM master

UCM sub 1

*(from Actors)*



*(from Actors)*

*(from Actors)*

*(from Actors)*

*(from Actors)*

:CampaignState=kActivating

Subscribe(CurrentStatus)

:READY

Activate(CampaignOrchestration)

:CurrentStatus = kActivating

Define ActivationMethod (Manifest or ActionMethod)

:CurrentStatus = kVerifying

CheckPackage Dependencies (SWCLDependencies)

**alt**

:CurrentStatus = kRollingBack

[U CM verify failing]

heck(False)

ling]

agregated verify fai

Vehicle C

[VSM

RollBack()

:Cam ling

paignState=k Cancel

lling

:TransferState=k Cance

:CurrentStatus = kRolledBack

Finish()

:CurrentStatus = kCleaningUp

CampaignState=kIdle

:

e

:TransferState=kIdl

**Figure C.7: Packages rollback by UCMs**

## Campaign reporting

OTA Client



*(from Actors)*

Driver Interface

*(from Actors)*

U CM master

*(from Actors)*

UCM sub 1

*(from Actors)*

UCM sub 2

*(from Actors)*

:CampaignState=IDLE

**par Checking UCM slaves states**

**opt Check UCM state**

Subscribe(CurrentStatus)

:IDLE

GetHistory(time StampGE, timeStampLT)

:GetHistory ReturnType

**opt**

Subscribe(CurrentStatus)

:IDLE

GetHistory(time StampGE, timeStampLT)

:GetHistory ReturnType

CampaignAggregation()

CampaignJudgement()

**opt Optional Action**

GetCampaignHistory(timeForm, timeTo)

:CampaignHistoryType

getCampaignHistory(timeFrom, timeTo)

:CampaignHistoryType

**Figure C.8: Campaign reporting to backend**

# Security Analysis of Installation and Update

This chapter presents a summary for the security analysis of the UCM. Some of the threats could not be addressed by specifying AUTOSAR requirements. The main rea- son 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:

## Securing Software Package

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

Information disclosure is another security threat category that might be applicable to Software Packages. Packages that contain sensitive information, such as intellec- tual 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 Software Update Packages is their freshness. An attacker may try to manipulate the system by downgrading the software via replaying an authen- tic but older Software Update 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.

## Securing Calls to UCM

UCM provides a very critical functionality in the platform that allows modifying appli- cations and platform components. In that sense, it is critical to prevent unauthorized access to UCM, meaning only legitimate callers should be allowed to reach the UCM service interface. This is primarily enforced in the communication layer supported by the Identity and Access Management. Additionally, the calls to the UCM interface 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 communica- tion, is additionally required.

Moreover, some API methods of the UCM interface returns sensitive information about the platform. This subset (GetSwClusterInfo, GetSwClusterChangeInfo, GetHistory, GetSwPackages) shall be protected against information disclosure and should only be reachable over a channel that provides confidentiality.

A similar reasoning is applicable for securing the communication between UCM Master and its clients. Regarding protection against information disclosure, GetSwClusterInfo, SwPackageInventory and GetHistory for UCM Master shall only be called over confi- dential channels.

## Suppressing Call to UCM

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

## Resource Starvation

According to the current specification, the available resources for transferring a Soft- ware 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. This scenario is also applicable to UCM Master when receiving data from its client. A similar case is possible for processing of Software Package, as the resources are only checked at the beginning but not re- served. In this regard, a solution could be to reserve the necessary resources for the Software Package transfer or processing from the beginning to prevent attacks aiming at such scenarios.

At the same time, reserving the resources might provide opportunity to the attacker in other scenarios. The specification allows transferring multiple Software Packages in parallel. Consequently, a misbehaving or compromised client can open unlimited number of transfer sessions causing UCM to run out of resources. To cope with this scenario, a threshold for the number of parallel transfer sessions can be defined.

## Zombie Sessions

The AUTOSAR specification does not enforce any expiry time for the established trans- fer sessions. As a result, the resources that are hold by an ongoing session will not

be released no matter how long time it takes. At the same time, in certain cases it may take a long time for larger software packages to be transferred to UCM or UCM Master, especially when they are received from external sources with weak connec- tivity on-the-fly. However, a timeout may be considered for such a transfer to prevent attackers from mounting denial of service attacks by long term allocation of resources.