

OCORA

Open CCS On-board Reference Architecture

Proof of Concept Configuration Management

Test Report

This OCORA work is licensed under the dual licensing Terms EUPL 1.2 (Commission Implementing Decision (EU) 2017/863 of 18 May 2017) and the terms and condition of the Attributions- ShareAlike 3.0 Unported license or its national version (in particular CC-BY-SA 3.0 DE).



Document ID: OCORA-TWS015-061

Version: 1.0

Date: 31.01.2025

Management Summary

As part of the OCORA project, SBB has developed a high-level Configuration Management (CM) concept for railway equipment. The goal of this Proof of Concept (PoC) was to validate the update process for on-board systems, using modified Selectron Systems components.

The PoC aimed to demonstrate the feasibility of applying updated Vehicle Configuration components to an on-board system, focusing on essential requirements while excluding complex off-board elements. The system comprised a Configuration Manager (CM) and two Building Blocks (BBs). Testing hardware included Selectron components, a Raspberry Pi, and associated peripherals.

A series of test cases evaluated the PoC's performance under various scenarios, including manual and automated update modes, failure scenarios, and activation of new configurations.

Key challenges identified during the PoC include:

- Synchronization and Long-Polling: The use of long-polling for communication between CM and BB-UMs resulted in delays and required repeated commands for successful updates.
- Transparency and Logging: A lack of comprehensive logging and status synchronization limited system traceability. Additional monitoring hardware, such as status indicators, would enhance visibility.

The PoC demonstrated a basic implementation of the configuration management process but highlighted critical gaps in transparency, synchronization, and logging. Selectron Systems provided SBB with the foundational files for potential internal development. However, SBB decided to not continue the configuration management activities under the OCORA collaboration. Future configuration specifications and concepts shall be investigated in EU-Rail.

Revision history

Version	Change Description	Initial	Date of change
0.2	▪ Initial draft	TF	02.12.2024
1.0	▪ Final version for OCORA Release R6	TF	31.01.2025

Table of contents

1	Introduction	7
1.1	Purpose of the document.....	7
1.2	Applicability of the document	7
1.3	Context of the document.....	7
2	Proof of Concept: Configuration Management	8
3	Performed test cases	10
3.1	Vehicle configurations.....	12
3.2	Observations.....	12
3.2.1	Case #02 Update of IEC-App SW of two BBs (Auto Mode of CM)	12
3.2.2	Case #04 Update of IEC-App SW of one BB (Auto Mode of CM)	13
3.2.3	Case #11 CM not available at start-up	13
3.2.4	Case #31 Activation of new Vehicle Configuration: In Safe State of IEC-Apps (Manual Mode of CM) Case #32 Activation of new Vehicle Configuration: In Safe State of IEC-Apps (Auto Mode of CM)	13
3.2.5	Case #33 Re-Activation of initial Vehicle Configuration: In Safe State of IEC-Apps 13	
3.2.6	Case #43 Failure identified during Verification of BB-Package by BB-UM after reception of Update Trigger	14
4	Identified and known issues and restrictions	14
4.1	Limited monitoring and logging capabilities	14
4.2	Synchronization and Long-Polling	14
4.3	IEC and MOS+IEC Update	15
4.3.1	IEC App Update only	15
4.3.2	MOS and IEC App Update together	15
5	Conclusion and further proceeding	15
6	Appendix	15

Table of figures

Figure 1	Overview of the On-Board system with CM and BBs [9]	8
Figure 2	Sequence of applying new components of a Vehicle Configuration to the CM [9]	9
Figure 3	Components of the PoC Test System [9].....	10
Figure 4	Screenshot during testing with the Terminals of the BB-UMs, the CM and the help function.	11

Table of tables

Table 1 – Overview test cases	11
Table 2 Available Vehicle Configurations [10]	12

References

Reader's note: please be aware that the numbers in square brackets, e.g. [1], as per the list of referenced documents below, is used throughout this document to indicate the references to external documents. Wherever a reference to a TSI-CCS SUBSET is used, the SUBSET is referenced directly (e.g. SUBSET-026). OCORA always reference to the latest available official version of the SUBSET, unless indicated differently.

- [1] OCORA-BWS01-010 – Release Notes
- [2] OCORA-BWS01-020 – Glossary
- [3] OCORA-BWS01-030 – Question and Answers
- [4] OCORA-BWS01-040 – Feedback Form
- [5] OCORA-BWS03-010 – Introduction to OCORA
- [6] OCORA-BWS03-020 – Guiding Principles
- [7] OCORA-BWS04-010 – Problem Statements
- [8] OCORA-TWS07-060_Configuration Management-Concept
- [9] OCORA-TWS15-061_Appendix_SRS PoC Maintainability_v1.1, Version 1.1, Selectron Systems
- [10] OCORA SRS PoC User Manual_v0.4_2024_04_24, Version 0.4, Selectron Systems

1 Introduction

1.1 Purpose of the document

The purpose of this document is to describe the results of the proof of concept of a demonstrator for concept of configuration management [\[8\]](#).

This document is addressed to experts in the CCS domain and to any other person, interested in the OCORA concepts for on-board CCS. The reader is invited to provide feedback to the OCORA collaboration and can, therefore, engage in shaping OCORA. Feedback to this document and to any other OCORA documentation can be given by using the feedback form [\[4\]](#).

If you are a railway undertaking, you may find useful information to compile tenders for OCORA-inspired CCS building blocks, for tendering complete on-board CCS systems, or for on-board CCS replacements for functional upgrades or life-cycle considerations.

If you are an organization interested in developing CCS on-board building blocks according to the OCORA design principles, the information provided in this document can be used as input for your development.

1.2 Applicability of the document

The document is informative. Subsequent releases of this document will be developed based on a modular and iterative approach, evolving within the progress of the OCORA collaboration.

1.3 Context of the document

This document is published as part of an OCORA Release, together with the documents listed in the Release Notes [\[1\]](#). Before reading this document, it is recommended to read the Release Notes. If you are interested in the context and the motivation that drives OCORA we recommend reading the Introduction to OCORA [\[5\]](#), the Guiding Principles [\[6\]](#), and the Problem Statements [\[7\]](#). The reader should also be aware of the Glossary [\[2\]](#) and the Question and Answers [\[3\]](#).

2 Proof of Concept: Configuration Management

As part of the OCORA project, SBB had developed a high-level Configuration Management concept [7] for railway equipment. This concept was the base for the specification of the PoC. The scope of the PoC, however, was to provide and demonstrate a realizable approach of publishing and applying components of a Vehicle Configuration to the On-Board system.

The detailed documentation for the technical implementation is described in [9].

The concept was simplified to serve the essential requirements for the PoC with focus on the On-Board side and without the sophisticated Off-Board aspects, such as the creation, merging and checking of vehicle configurations, the transfer from the wayside to the vehicle via mobile communication, etc. The following figure provides an overview and illustrates the application of the components of a Vehicle Configuration to the On-Board system consisting of the Configuration Manager and two Building Blocks (BBs).

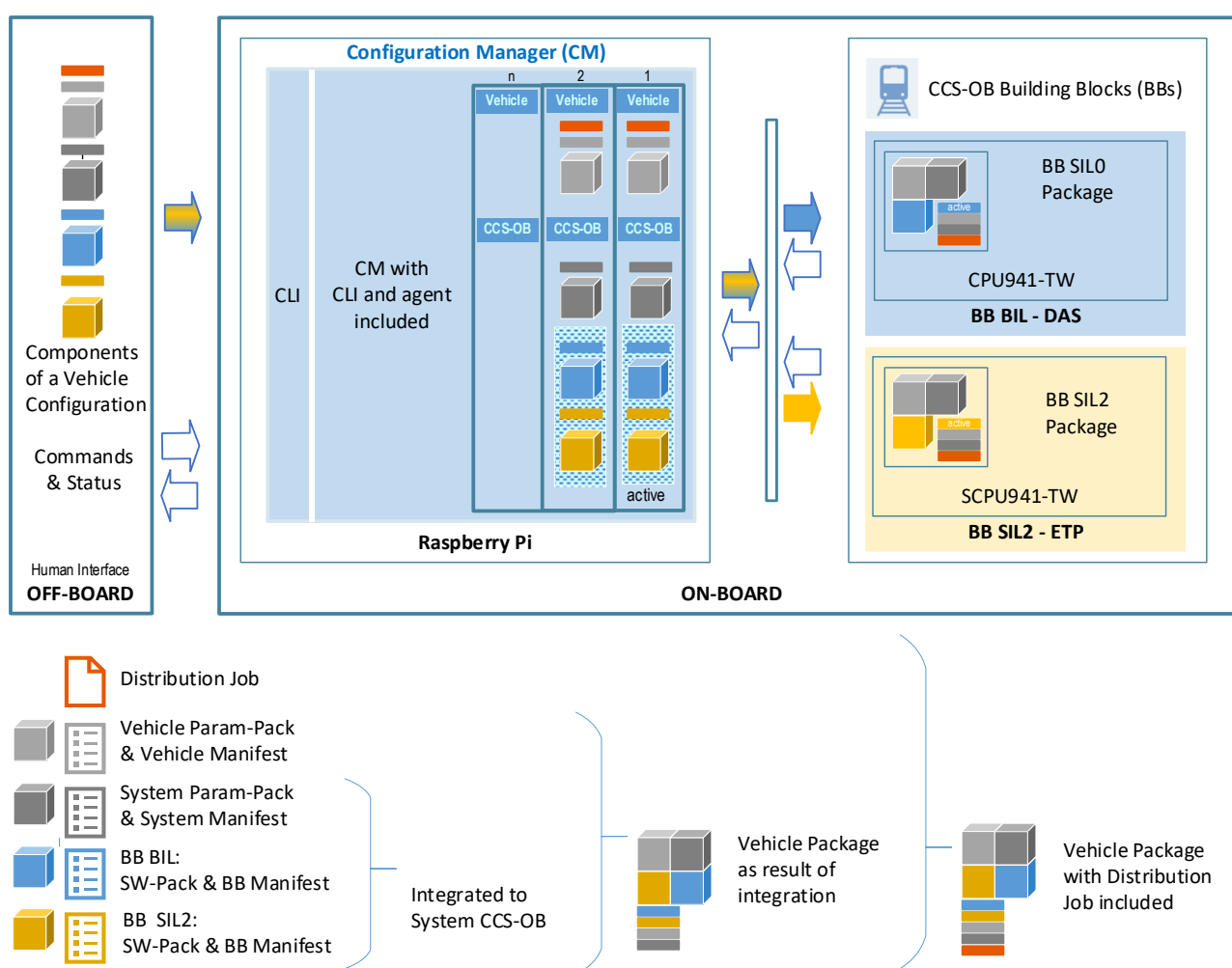


Figure 1 Overview of the On-Board system with CM and BBs [9]

The basic flow chart for the execution of the Update is shown in **Figure 2**.

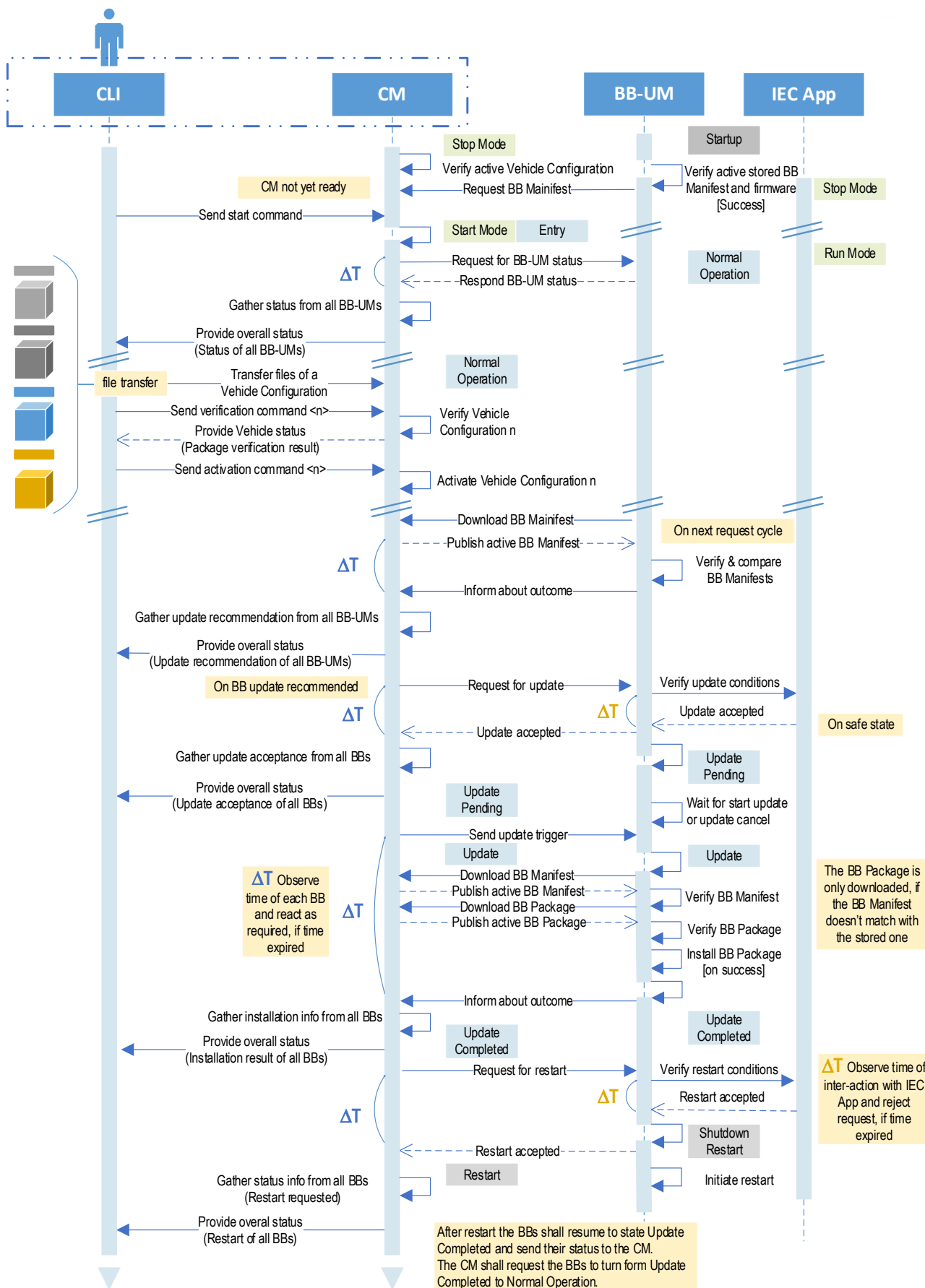


Figure 2 Sequence of applying new components of a Vehicle Configuration to the CM [9]

The test system for the PoC consists of on the front side of two SCPU941, a Raspberry Pi, a Selectron Switch and necessary accessories and cables to connect the components. A Power Supply 24V, a Selectron Security Gateway and a Power-On Button is additionally located at the backside.

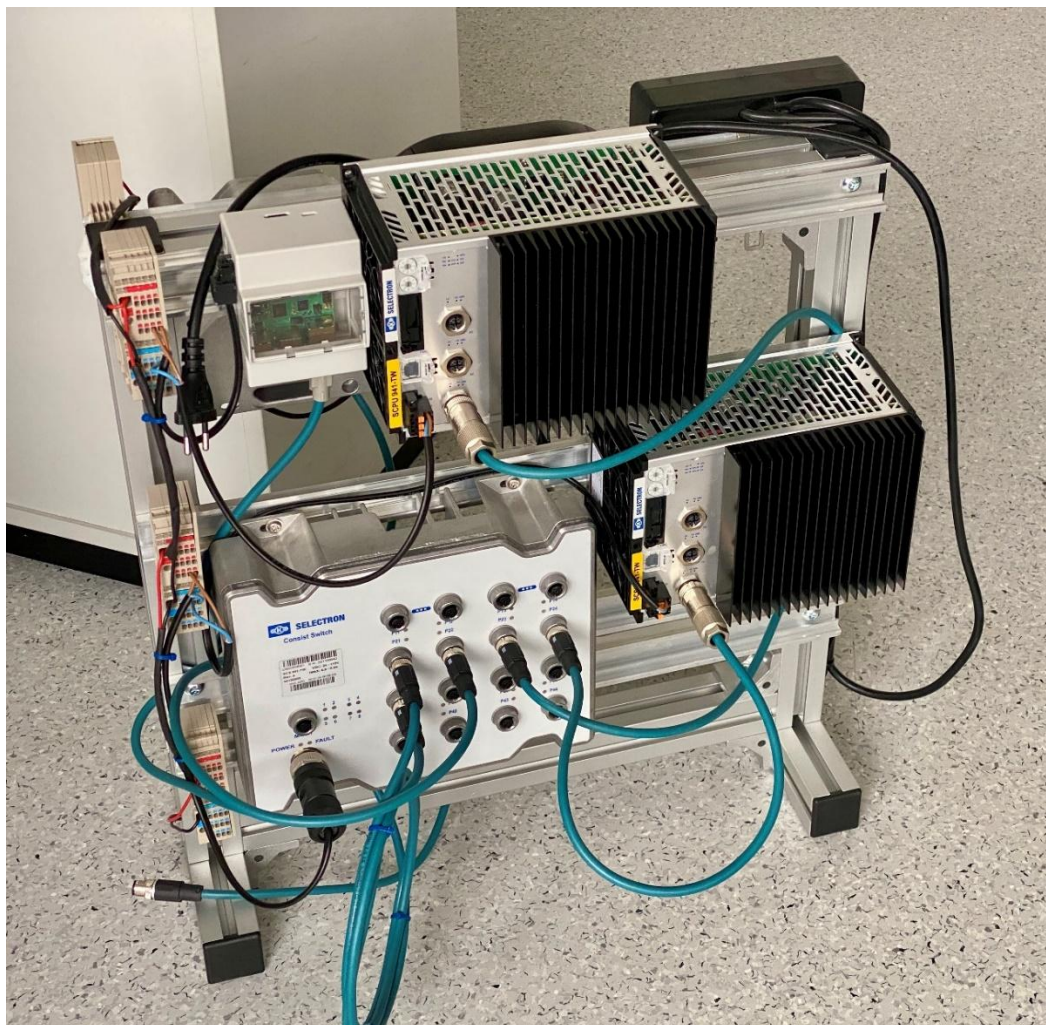


Figure 3 Components of the PoC Test System [9]

It was also discussed adding I/O elements to effectively demonstrate the exchange of IEC apps via CM and to easily monitor the state of the IEC apps, but this would have required additional effort, and was finally not realised.

3 Performed test cases

The tests were carried out and documented with the supplied implementation based on the discussed specification. As the implementation turned out to be more extensive than initially assumed, resources that were intended for the further development of the system after the initial execution of the tests had already been utilised for the first implementation. Adjustments to the implementation to improve the value of the PoC were therefore not carried out.

The following test cases were carried out and documented as part of the PoC.

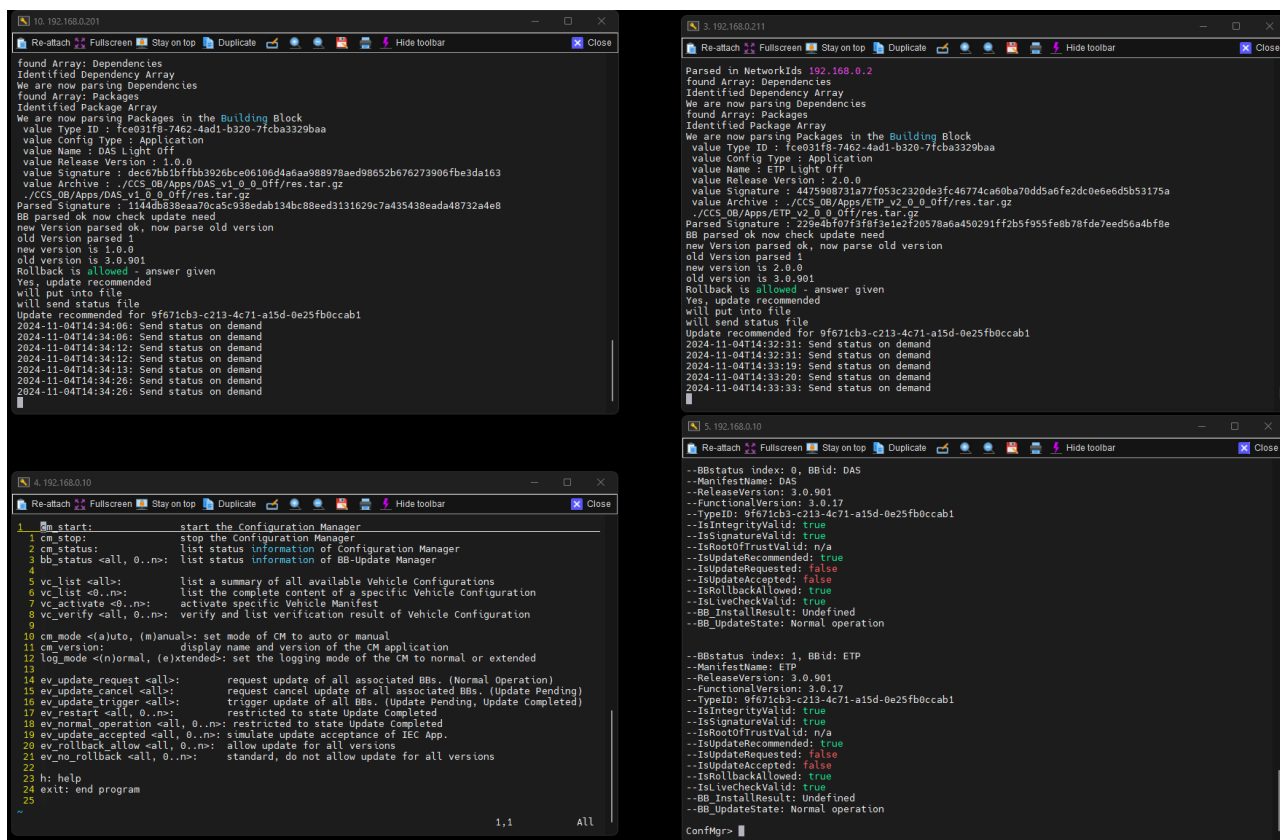
Test Case	Test case title
Case #01	Update of IEC-App SW of two BBs (Manual Mode of CM)
Case #02	Update of IEC-App SW of two BBs (Auto Mode of CM)
Case #03	Update of two BBs with MOS update (Manual Mode of CM) with restart
Case #04	Update of IEC-App SW of one BB (Auto Mode of CM)

Case #11	CM not available at start-up
Case #12	Restart of CM during update
Case #21	Timeout: Update request
Case #31	Activation of new Vehicle Configuration: In Safe State of IEC-Apps (Manual Mode of CM)
Case #32	Activation of new Vehicle Configuration: In Safe State of IEC-Apps (Auto Mode of CM)
Case #33	Re-Activation of initial Vehicle Configuration: In Safe State of IEC-Apps
Case #41	Failure identified during Verification of Vehicle Configuration by CM
Case #42	Failure identified during Verification of BB-Package by BB-UM after download from CM
Case #43	Failure identified during Verification of BB-Package by BB-UM after reception of Update Trigger

Table 1 – Overview test cases

The screenshot in **Figure 4** shows the terminal set-up used for testing: The two upper windows represent the terminals for the BB-UMs of the two BBs. The process data shown windows is not logged.

At the bottom right is the CLI of the CM. The process data shown in the window is stored in log-files. The possible usable commands from the help function are shown at the bottom left.



```

1. start: start the Configuration Manager
2. cm_stop: stop the Configuration Manager
3. cm_status: list status information of Configuration Manager
4. bb_status <all, 0..n>: list status information of BB-Update Manager
5. vc_list <all>: list a summary of all available Vehicle Configurations
6. vc_list <0..n>: list the complete content of a specific Vehicle Configuration
7. vc_activate <0..n>: activate specific Vehicle Manifest
8. vc_verify <all, 0..n>: verify and list verification result of Vehicle Configuration
9.
10. cm_mode <(a)uto, (m)anual>: set mode of CM to auto or manual
11. cm_version: display name and version of the CM application
12. log_mode <(n)ormal, (e)xtended>: set the logging mode of the CM to normal or extended
13.
14. ev_update_request <all>: request update of all associated BBs. (Normal operation)
15. ev_update_cancel <all>: request cancel update of all associated BBs. (Update Pending)
16. ev_update_trigger <all>: trigger update of all BBs. (Update Pending, Update Completed)
17. ev_restart <all, 0..n>: restricted to state Update Completed
18. ev_normal_operation <all, 0..n>: restricted to state Update Completed
19. ev_update_accepted <all, 0..n>: simulate update acceptance of IEC App.
20. ev_rollback_allow <all, 0..n>: allow update for all versions
21. ev_no_rollback <all, 0..n>: standard, do not allow update for all versions
22.
23. h: help
24. exit: end program
25.

```

```

--BBstatus index: 0, BBid: DAS
--ManifestName: DAS
--ReleaseVersion: 3.0.901
--FunctionalVersion: 3.0.17
--TypeId: 9f671cb3-c213-4c71-a15d-0e25fb0ccab1
--IsIntegrityValid: true
--IsSignatureValid: true
--IsRootOfTrustValid: n/a
--IsUpdateRequested: true
--IsUpdateAccepted: false
--IsRollbackAllowed: true
--IsLiveCheckValid: true
--BB_InstallResult: Undefined
--BB_UpdateState: Normal operation

--BBstatus index: 1, BBid: ETP
--ManifestName: ETP
--ReleaseVersion: 3.0.901
--FunctionalVersion: 3.0.17
--TypeId: 9f671cb3-c213-4c71-a15d-0e25fb0ccab1
--IsIntegrityValid: true
--IsSignatureValid: true
--IsRootOfTrustValid: n/a
--IsUpdateRequested: true
--IsUpdateAccepted: false
--IsRollbackAllowed: true
--IsLiveCheckValid: true
--BB_InstallResult: Undefined
--BB_UpdateState: Normal operation

```

Figure 4 Screenshot during testing with the Terminals of the BB-UMs, the CM and the help function.

3.1 Vehicle configurations

The following vehicle configurations were available for the tests

0) Vehicle Manifest v0 (0a/0b*): (IEC App only "900" TrackRunner)	BB 0 for DAS with IEC App v1.0.0 (light off) BB 1 for ETP with IEC App v2.0.0 (light off)
1) Vehicle Manifest v1 (0a/0b*): (MOS & IEC "900" TrackRunner)	BB 0 for DAS with MOS v3.0.900 BB 1 for ETP with MOS v3.0.900 BB 0 for DAS with IEC App v1.0.0 (light off) BB 1 for ETP with IEC App v2.0.0 (light off)
2) Vehicle Manifest v2 (10a/10b): (MOS & IEC "901" QuantumSpirit)	BB 0 for DAS with MOS v3.0.901 BB 1 for ETP with MOS v3.0.901 BB 0 for DAS with IEC App v1.5.1 (light on) BB 1 for ETP with IEC App v2.5.1 (light on)
3) Vehicle Manifest v3 (1/3*): (IEC App only "900" TrackRunner)	BB 0 for DAS with IEC App v1.1.0 (slow counter) BB 1 for ETP with IEC App v2.1.0 (slow run light)
4) Vehicle Manifest v4* (2/4*): (IEC App only "900" TrackRunner)	BB 0 for DAS with IEC App v1.1.0 (fast counter) BB 1 for ETP with IEC App v2.2.0 (fast run light)
5) Vehicle Manifest v5 (11a/11b*): (IEC App only "901" QuantumSpirit)	BB 0 for DAS with IEC App v1.3.1 (run light 11) BB 1 for ETP with IEC App v2.3.1 (run light 11)
6) Vehicle Manifest v6 (5/7*): (IEC App only "900" TrackRunner)	BB 0 for DAS with IEC App v1.3.0 (run light 11) BB 1 for ETP with IEC App v2.3.0 (run light 11)
7) Vehicle Manifest v7 (6/8*): (IEC App only "900" TrackRunner)	BB 0 for DAS with IEC App v1.4.0 (blinky 1) BB 1 for ETP with IEC App v2.4.0 (blinky 2)
8) Vehicle Manifest v8 (9a/9b*): (IEC App only "900" TrackRunner)	BB 1 for ETP with IEC App v1.5.0 (light on) BB 1 for ETP with IEC App v2.5.0 (light on)
9) Vehicle Manifest v9 (12a/12b*): (IEC App only "901" QuantumSpirit)	BB 0 for DAS with IEC App v1.6.1 (surprise) BB 1 for ETP with IEC App v2.6.1 (surprise)

Table 2 Available Vehicle Configurations [10]

The original Vehicle Configuration 4 has been modified so that only the Software of one IEC app (ETP) is updated when the basic configuration is Vehicle Configuration 3.

3.2 Observations

The anomalies identified are summarized below. These overlap with the restrictions in chapter 4.

3.2.1 Case #02 Update of IEC-App SW of two BBs (Auto Mode of CM)

Update in Auto Mode does not run automatically (except for the `ev_update_accepted` command) as expected. Command Update trigger is necessary. This command is not processed by the BB index 0 (DAS). Manual intervention through repetition of the commands is necessary. Change of the BB index 1 (ETP) to Normal Operation is only visible when a new request (`bb_status`) is commanded to the CM.

Evaluation

The cause lies in the application of the long-polling method for the communication between CM and the BB-UMs.

3.2.2 Case #04 Update of IEC-App SW of one BB (Auto Mode of CM)

The update is carried out on both BBs according to CM and CLI, although an update to the same version (for BB index 1) requires the 'Rollback allowed' function to be activated.

Additionally, it appears from the unlogged command files of the BB-UMs that the IEC App, which does not need to be updated, does not switch to Stop mode (or Safe State). However, this is suggested by the CLI messages. The states of the IEC Apps are not clearly visible."

Evaluation

The reason for this is that dependencies between BB have not yet been taken into account in the update process. The discrepancies in the status description of the BB-UM between the BB-UM and the CLI of the CM cannot be resolved.

3.2.3 Case #11 CM not available at start-up

The communication between the IEC-Apps and the respective BB-UM is not accessible. The states of the BB-UMs and the IEC-Apps is not logged without a started CM.

When the CM is started, a vehicle configuration must always be selected before the state of the BB-UM is displayed.

Evaluation

The fact that feedback from the BB-UMs can only be viewed with an activated configuration is related to the chosen implementation.

3.2.4 Case #31 Activation of new Vehicle Configuration: In Safe State of IEC-Apps (Manual Mode of CM) Case #32 Activation of new Vehicle Configuration: In Safe State of IEC-Apps (Auto Mode of CM)

The activation of the new Vehicle Configuration is possible even though the update cycle is already running. The update is not canceled by the CM.

Continuing the update cycle results in the software being installed in accordance with Vehicle Configuration 6, even though the initial check to see whether an update is necessary is missing.

Evaluation

From the project team's perspective, activation of a newer vehicle configuration during an update cycle should trigger a cancellation of this cycle or the communication between CM and BB-UM shall contain the respective versions to which commands and feedback messages relate.

3.2.5 Case #33 Re-Activation of initial Vehicle Configuration: In Safe State of IEC-Apps

The activation of the new Vehicle Configuration is possible even though the update cycle is already running.

The update is not interrupted by the CM.

The update is carried out according to CM and CLI, although an update to the same version requires the 'Rollback allowed' function to be activated.

In addition, it seems in the unlogged command files of the BB-UMs that the update is not carried out, but this is implied by the CLI messages. The states of the IEC Apps are not clearly visible.

Evaluation

The same statement applies to this case as to the two previous test cases.

It is also necessary that status messages within the system between the CM, BB-UM and IEC app elements are always unambiguous and, if feedback is provided, synchronized. Additional log functions are required for later traceability.

3.2.6 Case #43 Failure identified during Verification of BB-Package by BB-UM after reception of Update Trigger

Hashes for the published DAS manifest (packages and IEC-App) were manipulated before transmission to the BB-UM.

It can be seen from the BB-UM sequence that the update does not seem to be executed, despite the CLI suggesting otherwise. Here, BB index 0 (DAS) also runs through the regular sequence.

Evaluation

It is necessary that status messages within the system between the CM, BB-UM and IEC app elements are always unambiguous and, if feedback is provided, synchronized. Additional log functions are required for later traceability.

4 Identified and known issues and restrictions

4.1 Limited monitoring and logging capabilities

The limited monitorability of the system proved to be a fundamental challenge. Discrepancies between the information that could be viewed could not be clearly resolved.

Initially, only the CLI of the CM was available for monitoring the status. A log function only exists for the CLI.

The communication between CM and BB-UM was not visible. Access to the BB-UM terminal made it easier to track processes. A log function for this is not implemented.

However, a clear status of the IEC app could only be traced to a limited extent.

Additional hardware units, which indicate the status of the installed app version by means of light variations, would have been better. However, these were not procured.

4.2 Synchronization and Long-Polling

The implemented solution for communication between CM and BB-UMs is based on a web server approach with client server architecture. For synchronization purposes between CM and BB-UMs long-polling is used.

The long-polling approach does not guarantee 100% synchronization success between CM and BB-UMs without additional checks. Therefore, synchronization can only be guaranteed if, after a command has been

sent by the CM, the status response required by BB-UM is observed and the corresponding command is repeated until the correct response is received. In automatic mode, the CM performs this check automatically and repeats the command if necessary. In manual mode, the users must do this themselves.

In some cases, commands must be repeated several times due to this implementation. In addition, it was difficult to trace the cause of system reactions that did not occur.

4.3 IEC and MOS+IEC Update

4.3.1 IEC App Update only

If only IEC Apps are updated to the BB-UMs on the PoC test system, the behaviour in auto mode and manual mode is as specified at least if BB is updated. The system is not restarted just the IEC-App is set to run state again after returning to normal operation.

4.3.2 MOS and IEC App Update together

4.3.2.1 Do not yet use auto mode with MOS update

If both MOS and IEC Apps are updated to the BBUMs on the PoC test system, the CM should be used only in manual mode, since the timings for the CM not yet match with the long download and restart period.

4.3.2.2 Restart of IEC App too early

In case of a MOS and IEC update the system must be restarted. It depends on the switch S1 settings, if the IEC App is started automatically after restart or not. In the actual switch position of S1 the IEC App is directly started after the system restart. According to the specification the IEC App shall be restarted while returning to normal operation.

5 Conclusion and further proceeding

Based on the basic concept, a demonstrator was created after intensive discussions to harmonise the specification. The PoC demonstrated a basic implementation of the configuration management concept but highlighted critical gaps in transparency, synchronization, and logging.

Selectron Systems has provided the client SBB with the necessary basic files (source code, makefile, etc.) for a possible internal further development of the demonstrator.

As part of the System and Innovation Pillars of the ERJU approach, the basic concept of Configuration Management [8] was also included in the discussions and further developed in a separate approach. SBB decided to not continue the configuration management activities under the OCORA collaboration.

6 Appendix

Specification

PoC SBB Maintainability

Document identification:	OCORA SRS PoC Maintainability.docx
Authors:	Christian Dudka, Gerold Flaskaemper
Version:	1.1
Date:	24.04.2024
Classification:	
Release:	-

Content

1. Document Identification	5
1.1 History	5
1.2 Purpose and Scope	5
1.3 References	6
1.4 Abbreviations and definitions	6
1.4.1 Definition of a Manifest	7
1.4.2 Definition of a Configuration	7
1.4.3 Definition of a Building Block	7
1.4.4 Definition of a Package	7
2. Introduction and Overview	8
3. Required scenarios and use cases	10
3.1 Execute Normal Operation	10
3.1.1 Normal Operation on startup	10
3.1.2 Normal Operation in progress	10
3.1.3 The CM provides compliance of update package	10
3.2 Update	10
3.2.1 Transfer components of a new Vehicle Configuration to the CM	10
3.2.2 Activate new Vehicle Configuration	11
3.2.3 Start update process	11
3.2.4 Apply Update	11
3.2.5 Resume Operation	11
3.3 Rollback	11
3.3.1 Rollback triggered by the CM	11
3.4 Error handling	12
3.4.1 List of possible errors	12
3.4.2 The CM is not available on startup	12
4. Building Block requirement specification	13
4.1 Interfaces of the BB Update Manager	15
4.1.1 Configuration Manager interface	15
4.1.1.1 BB-UM get its BB Manifest	15
4.1.1.2 BB-UM responds the outcome of its BB Manifest comparison	15
4.1.1.3 BB-UM receives update request	15
4.1.1.4 BB-UM waits for update trigger in state Update Pending	15
4.1.1.5 BB-UM starts update	15
4.1.1.6 BB-UM responds with outcome of the update	16
4.1.1.7 BB-UM receives change to Normal Operation	16
4.1.1.8 BB-UM receives an update trigger in state Update Completed	16
4.1.1.9 BB-UM receives a restart request	16
4.1.1.10 BB-UM receives update cancel request	16
4.1.2 IEC App interface	17
4.1.2.1 The BB-UM gets status from IEC App run/stop	17
4.1.2.2 The BB-UM gets permission from IEC App for update	17
4.1.2.3 The BB-UM gets permission from IEC App for shutdown/restart	17
4.1.2.4 The BB-UM sends SW versions to the IEC App	17
4.2 Common requirements of the BB	17
4.2.1 BB-UM receives turn to error state request	17
5. Configuration Manager requirement specification	18
5.1 Interfaces of the Configuration Manager	20
5.1.1 File transfer to the CM	20
5.1.1.1 CM file transfer protocol	20
5.1.1.2 CM storage capacity	20
5.1.2 CLI command line interface	20
5.1.2.1 Start/ Stop the activity of the CM	20
5.1.2.2 Set active Vehicle Configuration	20

5.1.2.3	Verify active Vehicle Configuration on startup	20
5.1.2.4	Provide status of the Vehicle Manifest	20
5.1.2.5	Provide overall status of the participating BBs	21
5.1.2.6	Provide commands to trigger BB state transitions manually	21
5.1.2.7	Auto and manual mode of the CM	21
5.1.3	BB-UM interface	22
5.1.3.1	CM gathers current BB-UM status	22
5.1.3.2	CM synchronizes status of the BB-UMs	22
5.1.3.3	CM evaluates availability of all BB-UMs	22
5.1.3.4	CM provides BB Manifest	22
5.1.3.5	CM gathers update recommendation	22
5.1.3.6	CM sends update request	22
5.1.3.7	CM gathers update accepted	22
5.1.3.8	CM evaluates practicable update sequence	23
5.1.3.9	CM sends update trigger	23
5.1.3.10	CM provides BB Package	23
5.1.3.11	CM gathers update result	23
5.1.3.12	CM requests to restart the system	23
5.1.3.13	CM requests change to Normal Operation	23
5.1.3.14	CM requests re-update or rollback	23
5.1.3.15	CM request to cancel update	24
5.1.3.16	CM changes to error state	24
5.2	Common requirements of the CM	24
5.2.1	Status message definitions	24
5.2.2	CM/ BB-UM command definitions	25
5.2.2.1	Commands from BB-UM to CM	25
5.2.2.2	Commands from CM to BB-UM	25
5.2.3	CLI command definitions	25
5.2.3.1	CLI Operational commands	25
5.2.3.2	CLI Test commands	26
5.2.4	Command response	26
5.2.5	Logging of events	26
5.2.6	Displaying of status information	27
6.	Manifest requirement specification	29
6.1	Requirements	29
6.1.1	File format of a Manifest	29
6.1.2	Pattern of a Manifest	29
6.1.3	Mandatory and optional elements of the Manifest	29
6.1.4	Support of referenced files	29
6.1.5	Integrity	30
6.1.6	Identification and authentication	30
6.1.7	Compatibility and versions of different BBs	30
6.1.8	BB dependencies and update hierarchy	30
6.1.9	Parser capabilities	30
6.1.9.1	Reading information	30
6.1.9.2	Retrieving file paths	30
6.1.9.3	Checking syntax and semantic	30
6.1.9.4	Checking integrity	31
6.1.10	Config Items and Config Types in the Manifest structure	31
6.1.10.1	Dependency of Config Types	31
6.1.10.2	Nesting possibility of Config Types	31
6.1.11	Config Items and its Elements	32
6.1.12	Explanation of the Elements	32
6.1.12.1	Config Type	32
6.1.12.2	Type ID and Config ID	33
6.1.12.3	Name, Description, Supplier and Metadata	33
6.1.12.4	Release and Functional Version	33
6.1.12.5	Manifest	33

6.1.12.6	Archive	33
6.1.12.7	Network IDs	33
6.1.12.8	Systems	33
6.1.12.9	Building Blocks	34
6.1.12.10	Packages	34
6.1.12.11	Reference	34
6.1.12.12	Dependencies	34
6.1.12.13	Hash Type	34
6.1.12.14	Hash	34
6.1.12.15	Signature Type	34
6.1.12.16	Signature	34
6.1.13	Example JSON files	35
6.1.13.1	Example Manifest of a Vehicle Configuration	35
6.1.13.2	Example Manifest of a Building Block	35
6.1.13.3	Example Vehicle Manifest with nested Systems	37
7.	Scope of functionality and restrictions for the PoC	38
8.	Implementation	40
8.1	PoC Test System	40
8.2	Design decisions	41
8.2.1	Parser	41
8.2.2	Web Server	41
8.2.3	Intermediate test environment	41
8.2.4	Obsolete CLI commands	41
8.3	Implementation details	42
8.3.1	CM/BB-UM file handling	42
8.3.2	Long Polling mechanism	43
8.3.3	Package Config Types	43

Table of Figures

Figure 1:	Overview of the On-Board system with CM and BBs	8
Figure 2:	State chart of the BB Update Manager Operational Mode	13
Figure 3:	State chart of the BB Update Manager parent states	14
Figure 4:	State chart of the CM	18
Figure 5:	Sequence of applying new components of a Vehicle Configuration to the CM	19
Figure 6:	Hierarchical levels of the different Configurations / Manifests	29
Figure 7:	Dependency of Config Types	31
Figure 8:	Nesting possibilities of Config Types	31
Figure 9:	Config Items and Elements of a Manifest	32
Figure 10:	Components of the PoC Test System	40
Figure 11:	Sequence of CM/BB-UM file handling	42
Figure 12:	Sequence of the Long Polling mechanism	43

1. Document Identification

1.1 History

Output No.	Name	Date	Change in chapter	Reason for change
0.1	Christian Dudka	04.09.2023	All	Initial version
0.2	Christian Dudka	12.09.2023	All	Internal review
0.3	Christian Dudka	19.09.2023 20.09.2023	All 6	1 st review with SBB Manifest requirements added
0.4	Christian Dudka	21.09.2023	2, (3), 4, 5, 6	2 nd review with SBB, reworked, (only wording changed), CM state chart added, CM requirement for update sequence added
0.5	Christian Dudka Gerold Flaskaemper	09.10.2023	1 – 6 8	Internal review with infoteam Chapter “implementation” added
0.6	Christian Dudka Gerold Flaskaemper	24.10.2023	1-5 6	3 rd review with SBB, findings corrected Only JSON examples corrected
0.7	Christian Dudka	30.10.2023	4 – 6	4 th review with SBB, findings corrected, introducing REST for the CM/BB-UM interface, manual and auto mode of CM described, changed wording of interactions between BB-UMs and CM to consider client-server architecture, normal and extended logging described. Example for Vehicle JSON-Configuration refined.
0.8	Christian Dudka	08.11.2023	6	5 th review with SBB, findings for Manifest specifications corrected, new discussed Config Types and Elements added and described. JSON-examples updated.
1.0	Christian Dudka	15.11.2023	6	6 th review with SBB, usage of references in JSON-example refined. Specification (chapter 1 – 6) released for PoC.
1.1	Gerold Flaskaemper Christian Dudka	17.11.2023 24.11.2023 27.11.2023 30.11.2023 06.12.2023 18.01.2024 19.03.2024 10.04.2024 12.04.2024 18.04.2024 24.04.2024	6, 8 5.2 7 6 5.2.5 5.2.2 8 5.2.4 8.3 6 8.3.3 6.1.13 7 7 5.2	Minor change in Vehicle Manifest Logging of events and displaying of status information refined. Agreement with SBB for PoC restrictions mentioned in chapter 7. Reference using element Name instead of Description Displaying/logging of parsing results dependent of display mode added in table. Chapter added to summarize the exchange of commands between the CM and its BB-Ums There is no need to document the implementation explicitly since all essential information is already specified for the PoC– just design decisions shall be listed instead. Refinement of chapter command response. Introduce chapter implementation details for file exchange via web server and long polling. Config Type “Application” is introduced. See also chapter 8.3.3 for explanation. The JSON files are adapted to the conditions of the PoC. Items for scope and restrictions added. Items for scope and restrictions added. Complete BB status message

1.2 Purpose and Scope

The purpose and scope of this PoC specification shall be to provide and demonstrate a simple, realizable approach of the SBB Configuration Management concept [R3] of publishing and applying components of a *Vehicle Configuration* to a small On-Board system consisting of a *Configuration Manager* running on a Raspberry Pi and two CPUs.

1.3 References

No.	Name	Identification
[R1]	Abbreviations and Definitions	GI5100-01
[R2]	OCORA Glossary	OCORA-BWS01-020_Glossary.pdf
[R3]	OCORA Configuration Management Concept	OCORA-TWS07-060_Configuration-Management-Concept_V2.docx
[R4]	Explanation of RESTful API	restfulapi.net
[R5]		
[R6]		
[R7]		
[R8]		
[R9]		
[R10]		

1.4 Abbreviations and definitions

Refer to document "Abbreviations and Definitions" [R1] and [R2]. The following important abbreviations are listed redundantly for easy reference.

BB	Building Block
BB-UM	Building Block Update Manager
BIL	Basic Integrity Level
CCS	Control-Command and Signaling
CCS-OB	CCS On Board
CLI	Command Line Interface
CM	Configuration Manager
DAS	Driver Advisory System (BIL)
ETP	European Train Protection (SIL4)
ETCS	European Train Control System
IEC	International Electrotechnical Commission (referring to IEC 1131 – standard for process control software)
OCORA	Open CCS On-board Reference Architecture
OB	On-Board
PRAMSS	Performance, Reliability, Availability, Maintainability, Safety and Security
PTU-OS	Physical Train Unit Operation System
REST API	<p>REST stands for representational state transfer [R4]. A REST API (also known as RESTful API) is an application programming interface that conforms to the constraints of REST architectural style and allows interactions with RESTful web services. The constraints of REST includes</p> <ul style="list-style-type: none"> • Client-server architecture with requests managed through HTTP • Stateless client-server communication • Cacheable data that streamlines client-server interactions • Uniform interface between components • A layered system that organizes each type of server • Code-on-demand (optional)
SSH	Secure Shell network protocol

SIL	Safety Integrity Level
TCMS	Train Control and Management System
VC	Vehicle Configuration

1.4.1 Definition of a Manifest

A Manifest shall be a complete, clear, precise and approved human readable description of a Vehicle, System or Building Block Configuration using a standardized lightweight data-interchange format. A Manifest shall include references to dependent components and subordinate Manifests and shall contain elements for verification of identification, authentication and compatibility.

1.4.2 Definition of a Configuration

A Configuration shall be a description of all items (components) required to operate a physical instance of a Vehicle, System or Building Block. It shall include default values for parameters.

1.4.3 Definition of a Building Block

A Building Block is an entity, having standardized functionality, standardized PRAMSS requirements (including Tolerable Functional Failure Rate [TFFR], Safety Integrity Level [SIL] and Safety Related Application Conditions [SRAC]), standardized interfaces (on all OSI Layers) towards other Building Blocks and/or external systems. Building Blocks are exchangeable and migratable, without impacting other Building Blocks. Building Blocks may be purchased from different suppliers and capable of being integrated by a third party.

1.4.4 Definition of a Package

A Package shall consist of its Manifest and of all items (components) referenced by its Manifest. If other Manifests are referenced, the derived items shall be included as well. The integrity respectively folder and file structure specified by the Manifests shall be strictly respected. A package shall be creatable analogous to its Manifest(s) on each level: Vehicle, System and Building Block.

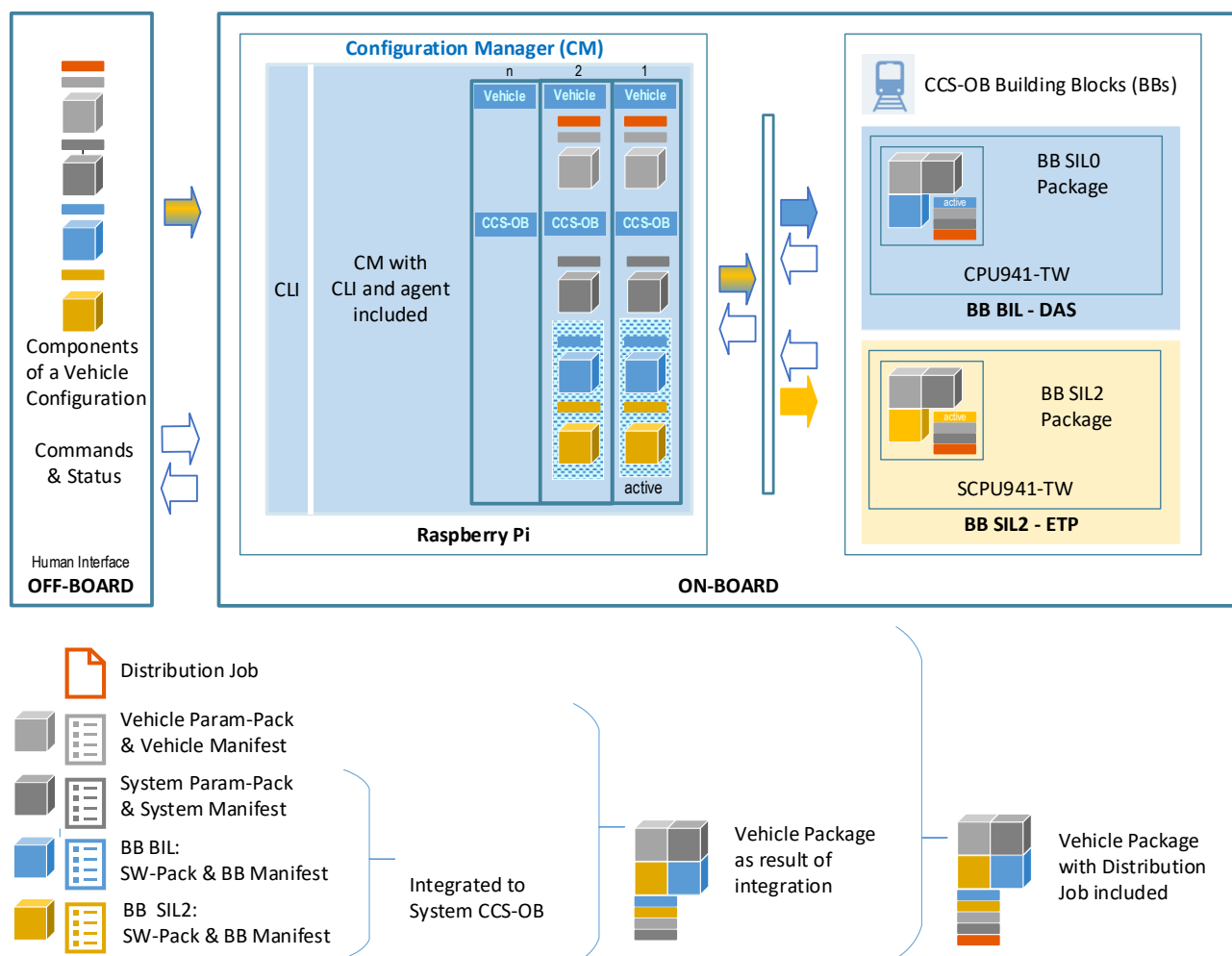
2. Introduction and Overview

As part of the OCORA project, SBB has developed a high-level Configuration Management concept for railway equipment. Goal of this specification is to demonstrate the proposed update process using modified Selectron components.

The OCORA Configuration and Management Concept [R3] of SBB shall be the base for this PoC maintainability specification. The scope of this PoC specification, however, shall be to provide and demonstrate a realizable approach of publishing and applying components of a *Vehicle Configuration* to the On-Board system.

The concept, [R3] figure 3, is simplified to serve the essential requirements for the PoC with focus on the On-Board side and without the sophisticated Off-Board aspects. The following figure provides an overview and illustrates the application of the components of a Vehicle Configuration to the On-Board system consisting of the Configuration Manager and just two Building Blocks (BBs).

Figure 1: Overview of the On-Board system with CM and BBs



A Vehicle Configuration shall contain its Manifest and a Param-Pack. For the PoC the Distribution Job is not considered and replaced by an activation command. The Param-Pack includes data necessary for the integration process. All referenced components in the Manifest shall belong to

the Configuration. These shall be basically the subordinate Building Blocks (BBs) each with its own subordinate Manifest and SW-Pack. A SW-Pack contains all necessary signed firmware parts and configurations, that are correctly parameterized for a successful installation and operation on a BB. Optionally the BBs can be grouped to logical entities e.g. BB BIL DAS and BB SIL2 ERP to System CCS-OB. See chapter 1.4 for definitions of Manifest, Configuration and Package.

For the PoC the Off-Board system shall be simplified and reduced to an operator sending commands to and retrieving status information from the On-Board system via command line (CLI). The On-Board system shall consist of the Configuration Manager (CM) and the Building Blocks (BBs). The agent and repository according to [R3] figure 3 shall be integrated to the CM. The BBs shall be just represented by two CPUs: one CPU941-TW dealing with a BB BIL e.g. for DAS and another SCPU941-TW dealing with a BB SIL2 e.g. for ETP.

The Vehicle Configuration with its referenced components shall be organized as filesystem with a folder file structure on the Configuration Manager (CM). The components of a Vehicle Configuration shall be exchangeable with a file transfer protocol. Different Vehicle Configurations shall be storable on the CM. On top level of the Vehicle Configuration the Vehicle folder shall be located and on bottom level the folder(s) for the Building Block(s). It shall be optional to group the BBs to logical coherent entities in intermediate folders.

The CM shall wait until it is requested by a BB to provide the correct BB Manifests of the Vehicle Package to the CPUs: In case of Normal Operation the BB Manifest with BIL integrity to the CPU941-TW and the BB Manifest with SIL2 integrity to the SCPU941-TW. Thereby the result of every interaction shall be sent to the CM and stored in a logfile. The content of the logfile shall be readable by command via CLI.

In the following chapters the use cases for the PoC and requirements for the CM, BBs and the Manifest are described in detail.

3. Required scenarios and use cases

In the interaction of Manifests, Configuration Manager (CM) and Building Blocks (BB) the PoC shall include the following scenarios “Normal Operation, Update, Rollback and Error Handling” demonstrated by the corresponding use cases. The basic idea shall be that the CM is the master of the update process of the BBs and has complete control over it. The CM shall be able to request for the actual status of the BBs in order to re-initiate synchronization between them.

3.1 Execute Normal Operation

3.1.1 Normal Operation on startup

On startup after switching to the parent state Operational Mode each BB shall request for the actual BB Manifest. The received BB Manifest shall be verified and compared with its own BB Manifest. If the CM is not available, the verification and comparison shall be skipped. Normal Operation shall be proceeded, if the stored BB Package and BB Manifest on the CPU are valid. If a BB Manifest from the CM was received, the BB informs the CM about its status of conformity with the active configuration. The actual Vehicle Configuration with status of all referenced BB participants including life checks shall be stored and available to be queried.

3.1.2 Normal Operation in progress

In Normal Operation the BB shall keep requesting the CM periodically for new BB Manifests. The received BB Manifest shall be verified and compared with its own BB Manifest. If the CM is not available, the verification and comparison shall be skipped. If a BB Manifest from the CM was received, the BB shall inform the CM about its status of conformity with the actual configuration. Thereby the CM assures that all BBs are periodically checking for their current Manifest (life check).

3.1.3 The CM provides compliance of update package

After startup and after every periodical request for BB Manifests the CM shall gather the outcome of all participating BBs and provide information about compliance of the active Vehicle Configuration including hierarchical structure, files, versions, and feedback from the BBs (status of conformity with the actual configuration on the CPUs).

3.2 Update

The process of applying a new BB Configuration, started by activating of a new Vehicle Configuration, is described in Figure 2.

3.2.1 Transfer components of a new Vehicle Configuration to the CM

Via file transfer new components of a Vehicle Configuration shall be stored for the CM. The CM shall verify by command the syntax, semantic and integrity of the components with the help of the included Vehicle Manifest and shall provide the result of the verification. The activation of a specific Vehicle Configuration shall be done by command. The activation shall be a replacement of the Distribution Job.

3.2.2 Activate new Vehicle Configuration

Once the new components referenced by the Vehicle Configuration have been activated, the CM shall respond with the correct new BB Manifest to all requests of the BBs. The BB verifies compliance and authentication of its new BB Manifest and informs the CM about its status of conformity with the new configuration including its recommendation for an update.

3.2.3 Start update process

The CM gathers the status of all BBs in a particular group and decides whether an update can be started. If an update is required, the CM requests all BBs to stop operational activities and proceed to state Update Pending. Each BB checks with the help of the IEC Application, if it is safe to stop functional operation and proceed with the update. If all conditions are met, the BB accepts the request and stops all operations. In any case the CM is informed whether the request is accepted or declined. The CM gathers the status of all BBs, repeats declined requests and waits for all to accept the update request.

Once all BBs are ready to start the update process, the CM starts the update by sending an update trigger to all BBs.

3.2.4 Apply Update

Immediately after the BB has received the update trigger and turned from state Update Pending to state Update, it requests the BB Package addressed by the new BB Manifest from the CM. Once the download of the BB Package is completed, the BB verifies its integrity, identity and authentication. On success the new BB Package with its SW-Packs including firmware and parameters are installed. The BB informs the CM about its status of conformity with the actual installation and turns in any case to state Update Completed. The progress of the update of each participating BB and the overall progress of all participants according to the Vehicle Configuration shall be provided.

3.2.5 Resume Operation

Once the BB changed to state Update Completed, it shall wait for further instructions from the CM: either changing to Normal Operation, initiating a system shutdown/ restart or an eventual rollback. In case of a restart the BB shall resume its previous update complete state and inform the CM about its actual status.

3.3 Rollback

3.3.1 Rollback triggered by the CM

The rollback possibility is part of the update process and allows the system to switch back to the previous BB Package in case the installation of a new BB Package has failed. Therefore, in the state Update Completed the BB shall inform the CM about the update result. If the installation of a new BB Package has failed, the CM shall offer the possibility to force either a re-update of the new BB Package or to force a rollback to the previous BB Package. The rollback is done by setting the previous components of the Vehicle Package as the active one and by retriggering the update process.

The use case of a rollback of an “old” device from stock, where the original Configuration is unknown by the CM, is not part of the PoC.

3.4 Error handling

3.4.1 List of possible errors

The following errors shall be handled:

- Missing BB Manifest
- Conformity of BB Manifest failed
- Integrity of BB Manifest failed
- Identity of BB Package failed
- Authentication of BB Package failed
- Installation of new BB Package failed
- Rollback failed

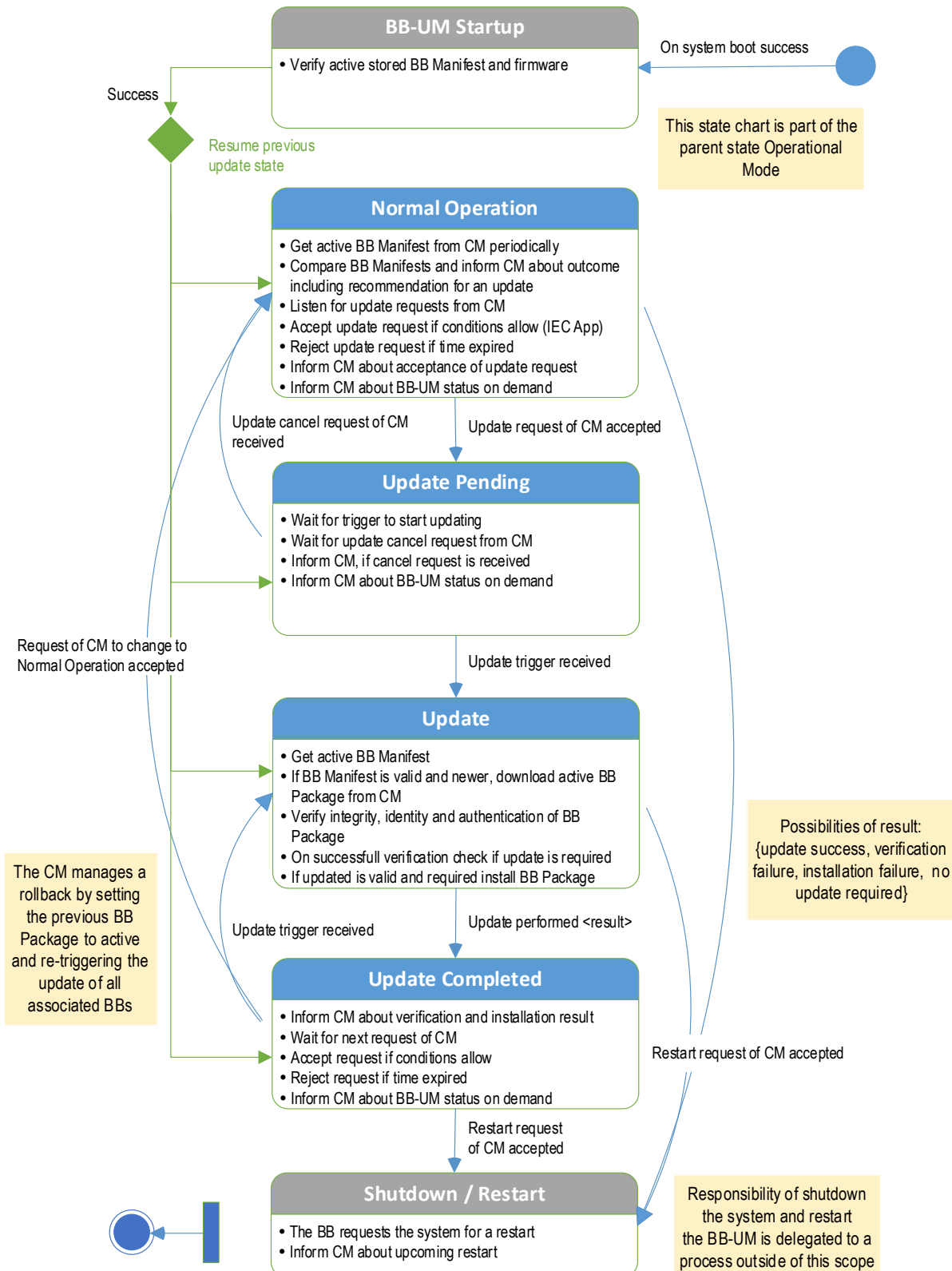
3.4.2 The CM is not available on startup

If the CM is not available on startup and the BB has already stored a valid and authentic BB Configuration, Normal Operation shall proceed and synchronization of its BB Manifest with the CM is done later on when the next periodical check is executed.

4. Building Block requirement specification

Figure 2: State chart of the BB Update Manager Operational Mode

The state chart of the BB-UM Operational Mode illustrates update process and interaction with the Configuration Manager (CM).



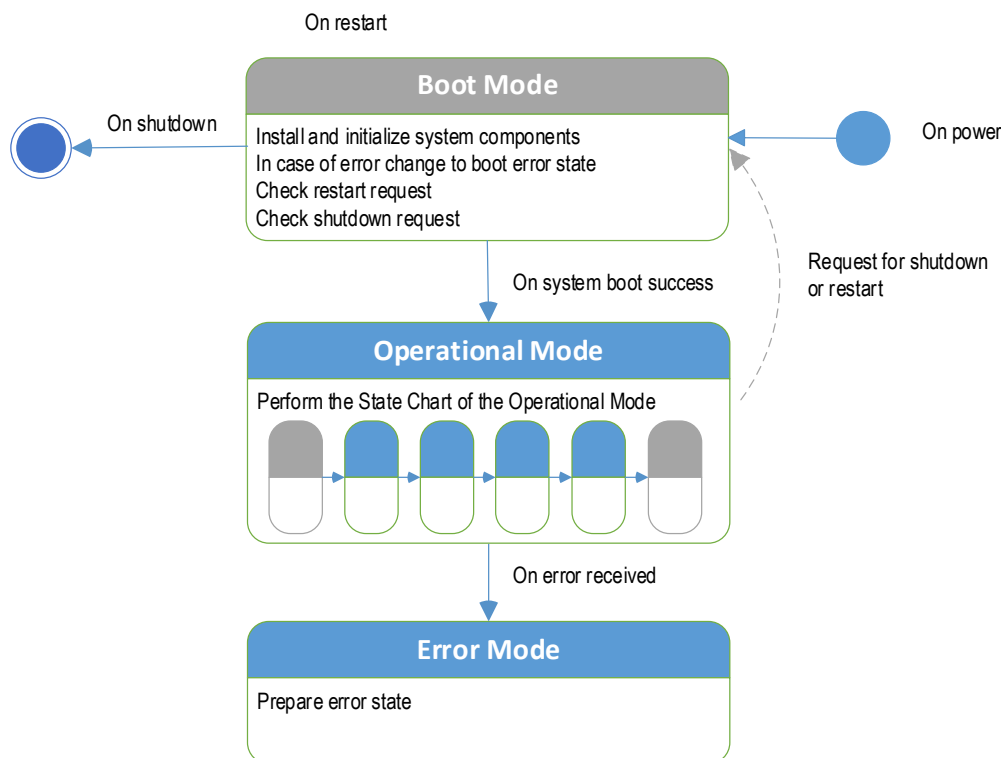
The state chart of the BB-UM above illustrates the four update states Normal Operation, Update Pending, Update and Update Completed. Moreover, there is a startup entry and a shutdown/restart exit state. The sequence is part of the parent state Operational. Another parent state shall be Error. Since the CM shall be able to request for a restart, the Update states shall be memorized and resumed on the subsequent startup. Therefore, it shall be in the responsibility of the CM to request for most state transitions. Startup and shutdown (program termination) is in the responsibility of a process outside the illustrated scope.

The verification of the BB Package on startup includes checking the BB Manifest for identity and authenticity and executing the signature verification of the firmware.

Whenever after a CM request the BB response depends on a condition that must be fulfilled in the overall system (IEC application), the BB shall check the response time from the overall system to be able to reject the CM request after a defined time interval.

Figure 3: State chart of the BB Update Manager parent states

The state chart of the BB-UM parent states illustrates the interaction of the states Boot Mode, Operational Mode, and Error Mode.



The Boot Mode is not part of the BB-UM but responsible for executing shutdown or restart requests. On installation success the BB-UM shall be started in Operational Mode. In case of an error the BB-UM changes to Error Mode where it shall remain until the system receives a restart or shutdown request.

4.1 Interfaces of the BB Update Manager

The firmware part of a BB that is responsible for the update process e. g. in the CPU shall be named BB Update Manager (BB-UM).

4.1.1 Configuration Manager interface

As the BB-UM is the counterpart of the CM it reflects almost all requirements. The communication between the CM and its participating BB-UMs shall use a *RESTful* architecture style approach based on the HTTP protocol [R4].

4.1.1.1 BB-UM get its BB Manifest

The BB-UM shall get on startup and later on in state Normal Operation periodically its BB Manifest from the CM. If the CM is not available, the BB-UM shall proceed without an error.

4.1.1.2 BB-UM responds the outcome of its BB Manifest comparison

After a BB Manifest was downloaded from the CM, the BB-UM shall compare the received BB Manifest with its current configuration (e.g. software versions, configuration hashes). The verification result, syntax, integrity, identity, authenticity and recommendation for an update shall be responded to the CM as status message.

4.1.1.3 BB-UM receives update request

Depending on the outcome and recommendation for an update the CM shall request the BB-UM to perform an update. The BB-UM therefore shall receive the request and verify the conditions for acceptance. The update permission shall be requested from the IEC Application with safety level integrity. Once the acceptance is available, the BB-UM shall change from state Normal Operation to state Update Pending and inform the CM that the update request was accepted. If an acceptance of the update request is not available within a certain time, the BB-UM shall send an update rejected to the CM. It shall be in the responsibility of the CM to retry the request.

4.1.1.4 BB-UM waits for update trigger in state Update Pending

After the BB-UM has accepted an update request from the CM, it waits in the state Update Pending for an update trigger. The trigger shall be sent by the CM when the conditions are fulfilled.

4.1.1.5 BB-UM starts update

After the BB-UM has received an update trigger, the BB-UM shall change from state Update Pending to state Update and download initially the active BB Manifest *). After the BB Manifest has been downloaded, the BB-UM verifies the update condition. If the downloaded BB Manifest doesn't match with its installed configuration the BB-UM shall download its BB Package (including SW-Pack with firmware and parameters). The gathered BB Package shall be verified by the BB-UM referring to syntax, integrity, identity and authenticity.

*) In this state the BB-UM has already downloaded an active BB Manifest from the CM. The initial repetition of downloading the BB Manifest is necessary, because a transition to state Update shall be possible not only from state Update Pending but as well from state Update Completed. In case of a rollback the CM has changed the active BB Manifest in the meantime.

4.1.1.6 BB-UM responds with outcome of the update

If the verification of the BB Package was successful, the BB Package shall be installed. In case of successful installation or failure the BB-UM shall change from state Update to state Update Completed and send the outcome of the update as status message to the CM. It shall be in the responsibility of the CM to react accordingly and send a corresponding state transition command to the BB-UM.

Possible outcomes of the update shall be: update successful, verification failure, installation failure, no update required.

4.1.1.7 BB-UM receives change to Normal Operation

Once an update is successfully completed, the BB-UM shall accept a change from state Update Completed to state Normal Operation from the CM, if the conditions allow this transition. The BB-UM shall inform the CM via status message about the acceptance of the request.

4.1.1.8 BB-UM receives an update trigger in state Update Completed

Once an update is finished and the BB-UM has entered state Update Completed, the BB-UM shall accept an update trigger from the CM, if the conditions allow this transition. The BB-UM shall inform the CM via status message about the acceptance of the request. If the request is accepted, the BB-UM shall change from state Update Completed to state Update again.

It shall be controlled by the CM, if a rollback to the previous configuration or a retrial of the present configuration shall be executed. The CM shall therefore set the corresponding Vehicle Configuration to active before it sends the start update trigger.

4.1.1.9 BB-UM receives a restart request

The BB Update Manager shall accept a restart request from the CM in the states Normal Operation, Update and Update Completed, if the conditions allow this transition. The reset permission shall be requested from the IEC Application with safety integrity. The BB Update Manager shall inform the CM via status message about the acceptance of the request before the request is delegated to the system.

4.1.1.10 BB-UM receives update cancel request

Once the update request was sent to the BB Update Manager, the BB shall wait for the start update trigger. In the state Update Pending the BB Update Manager therefore shall accept a cancel request from the CM and change to state Normal Operation. The BB Update Manager shall inform the CM via status message about the acceptance of the request.

4.1.2 IEC App interface

For the PoC the number of vPLCs on the xCPU941-TW shall be reduced to just one vPLC. The IEC Application is running in a safety integrity part of the system and shall therefore be queried for safety relevant permissions.

4.1.2.1 The BB-UM gets status from IEC App run/stop

The BB-UM shall have the possibility to get the status from the IEC App. If the application is stopped, an update request from the CM forwarded from the BB-UM to the IEC App cannot be answered. In this case the BB-UM shall send an update rejected to the CM. It is up to the CM to retry the update request.

4.1.2.2 The BB-UM gets permission from IEC App for update

After the BB-UM receives an update request from the CM, it shall request the IEC App for update permission. The IEC App shall signal the permission to the BB-UM if the required conditions are fulfilled. For the PoC a simple approach shall be considered. If an acceptance of the update request is not available within a certain time, the BB-UM shall send an update rejected to the CM. It shall be in the responsibility of the CM to retry the request.

4.1.2.3 The BB-UM gets permission from IEC App for shutdown/restart

After the BB-UM receives shutdown/ restart request from the CM, it shall request the IEC App for shutdown/restart permission. The IEC App shall signal the permission to the BB-UM if the required conditions are fulfilled. For the PoC a simple approach shall be considered. If an acceptance of the shutdown/ restart request is not available within a certain time, the BB-UM shall send a restart rejected to the CM. It shall be in the responsibility of the CM to retry the request.

4.1.2.4 The BB-UM sends SW versions to the IEC App

The BB-UM verifies its stored BB Manifest and BB Package and shall send the versions and status of the components to the IEC App.

4.2 Common requirements of the BB

4.2.1 BB-UM receives turn to error state request

Upon fatal error of the system the BB-UM shall accept a turn to error state request. If an exception in the system is detected that forces the safe state of the device, the entry process into the safe state of the system shall also inform the BB-UM to turn to its safe error state.

5. Configuration Manager requirement specification

Figure 4: State chart of the CM

The state chart of the CM illustrates the interaction with the CLI and the state machine of the BB-UM.

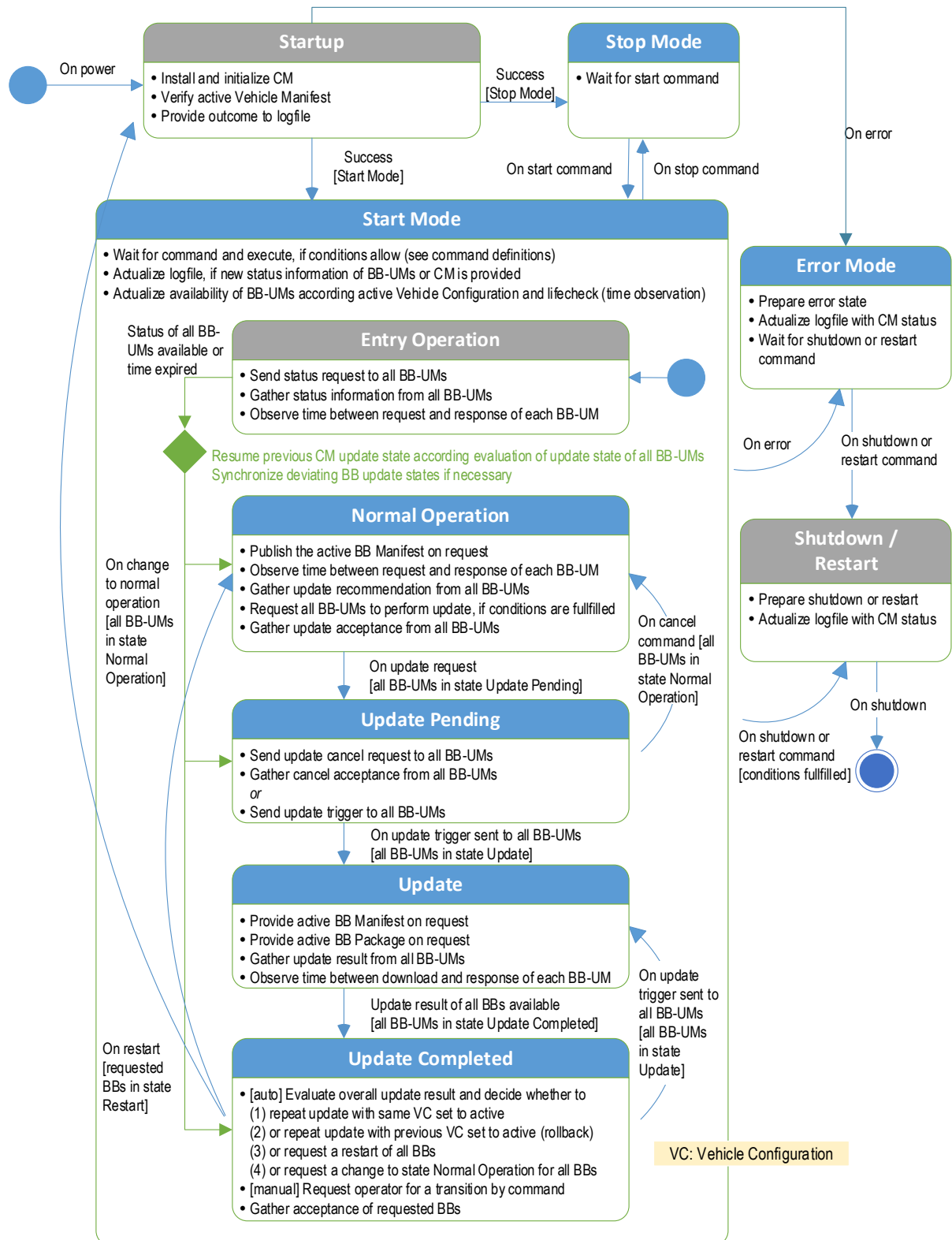
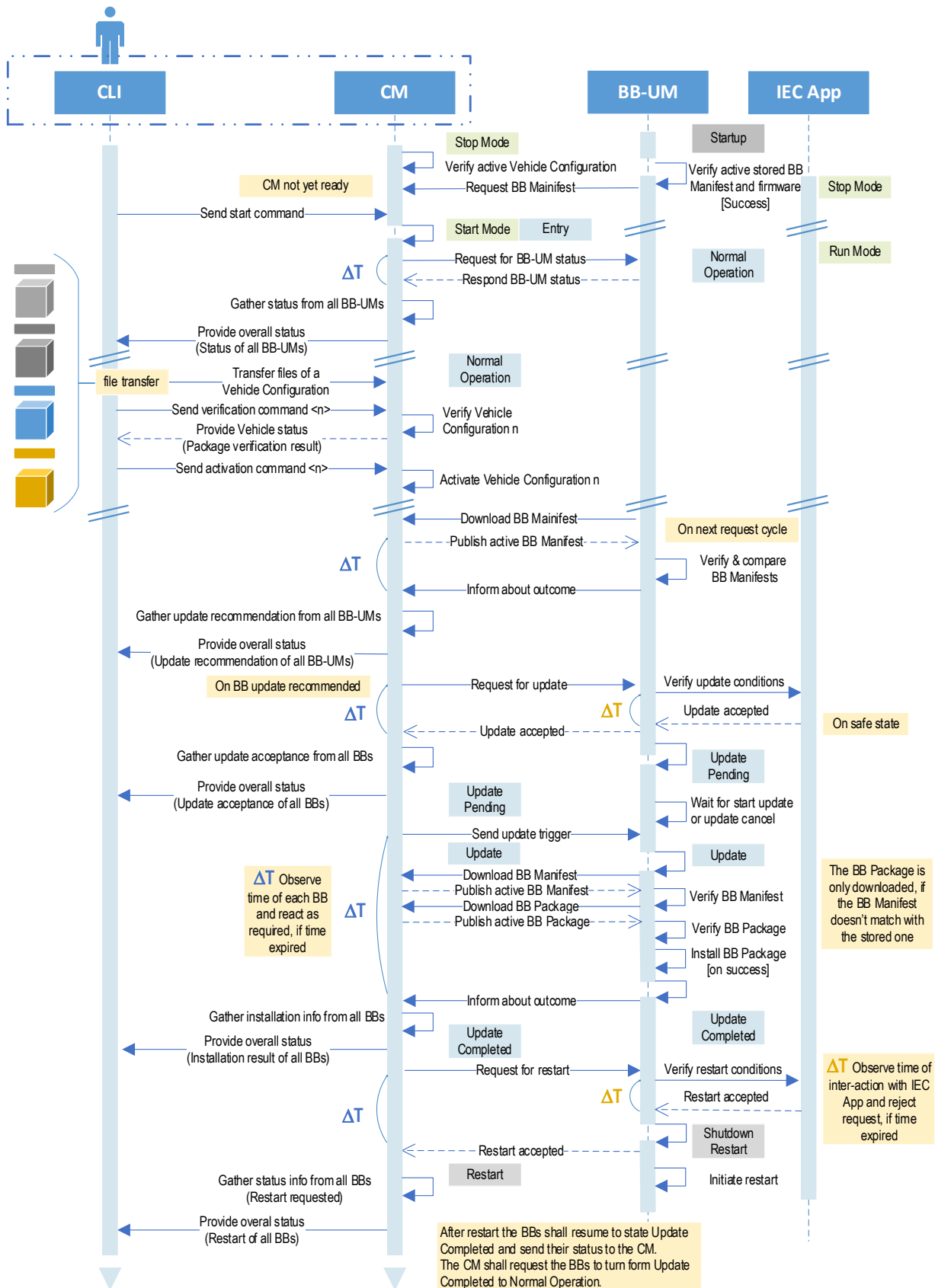


Figure 5: Sequence of applying new components of a Vehicle Configuration to the CM



The sequence above illustrates the context of the different participants while activating a new vehicle configuration of the CM according to the described use cases. Only the success sequence without error handling is considered.

5.1 Interfaces of the Configuration Manager

For the Configuration Manager (CM) the following interfaces shall be present:

5.1.1 File transfer to the CM

5.1.1.1 CM file transfer protocol

Via SSH a new Vehicle Configuration with its folders, files and items shall be transferable to the CM.

5.1.1.2 CM storage capacity

The CM shall have at least a capacity to store ten Vehicle Configuration with all its folders, files and items.

5.1.2 CLI command line interface

The CLI is part of the agent – for the PoC both included in the CM. The protocol for exchanging data between a human operator and the CM shall be SSH.

5.1.2.1 Start/ Stop the activity of the CM

The activity of the CM shall be controllable by a stop respectively a start command. The start/ stop state shall be stored persistently and resumed on startup. If set to stop, the CM shall not request or respond to the connected BBs. On set to start the CM shall synchronize with the BBs automatically by requesting for their status.

5.1.2.2 Set active Vehicle Configuration

Since the CM shall store a few Vehicle Configurations, it shall be possible to set the active configuration by command via CLI. The index of the activation shall be stored persistently and shall be resumed on restart.

5.1.2.3 Verify active Vehicle Configuration on startup

On startup the active Vehicle Configuration shall be verified referring syntax, integrity, identity and authenticity.

5.1.2.4 Provide status of the Vehicle Manifest

The status of a stored Vehicle Configurations of the CM shall be evaluated and queried by command. Since each Vehicle Configuration includes referenced BB Manifests, the status summarize the verification check of all depending Manifests referring to integrity, identity and authenticity. Moreover the status shall list name, version and the verification check of each dependent Manifest separately. The referenced firmware and parameter packages of each Manifest shall also be listed including names, versions and verification checks of the referenced files.

5.1.2.5 Provide overall status of the participating BBs

Since a started CM gathers status information from the connected BBs and stores it into a logfile, the overall status shall be provided by command. See BB status message definition.

5.1.2.6 Provide commands to trigger BB state transitions manually

Via CLI it shall be possible to send commands to the CM that are forwarded to the BBs to influence the state transitions of the BB-UMs manually. See Command list definitions.

5.1.2.7 Auto and manual mode of the CM

The CM shall be set by command either to auto or to manual mode. The two modes shall have an effect on how the different state transitions of the CM shall be executed:

Event	Auto mode	Manual mode
On startup of the CM:	The CM shall be started automatically.	The CM shall be started by a start command.
On all associated BB-UMs have provided their update state after startup:	If not all BB-UMs are in the same state, the CM synchronizes the states of the BB-UMs automatically by either changing all to state Normal Operation or state Update.	If not all associated BB-UMs are in the same state, the states of the BB-UMs are listed and the user is prompted to synchronize the states by command.
On all associated BB-UMs have provided their update recommendation:	The CM shall request all associated BB-UMs for an update automatically.	The CM shall prompt to send an update request. The update request is sent by an update request command.
On all associated BB-UMs have accepted an update request:	The CM shall trigger an update of all associated BBs automatically.	The CM shall prompt to send an update trigger or update cancel. The update trigger is sent by an update trigger command.
On one or more associated BB-UMs have rejected an update request:	The CM shall request all associated BB-UMs for an update again automatically. If the repetition was not successfully, the CM shall prompt for a decision by user interaction.	The CM shall prompt for a decision by user interaction.
On all associated BB-UMs have completed their update:	The CM shall evaluate the overall update result of all associated BB-UMs and decides, whether to trigger an update (retrial or rollback *)), restart the system or change to state Normal Operation automatically. *) Rollback shall be done by setting previous VC to active in advance.	The CM shall list the update result and prompt for an user interaction, either to repeat the update, execute a rollback *), restart the system or change to state Normal Operation. *) Rollback shall be done by setting previous VC to active in advance.
On exception in auto mode:	If the internal logic of the CM is not in a position to decide the next step or transition automatically, it shall prompt for a decision by user interaction.	none
On fatal error of the CM:	The CM shall prompt for a shutdown or restart command.	The CM shall prompt for a shutdown or restart command.

5.1.3 BB-UM interface

As the BB-UMs are the counterpart of the CM, they reflect almost all interface definitions. The communication between the CM and its participating BB-UMs shall use a *RESTful* architecture style approach based on the HTTP protocol [R4].

5.1.3.1 CM gathers current BB-UM status

After the CM is started and changed to Entry Operation it shall gather the status of all BB-UMs by request and store a summary of the outcome that can be queried by command.

5.1.3.2 CM synchronizes status of the BB-UMs

The CM shall evaluate the outcome of the status of all BB-UMs on Entry Operation and decide to which CM state it shall enter. In case not all BB-UMs have the same state, the CM shall evaluate, if a synchronization of the BB-UM states is required.

5.1.3.3 CM evaluates availability of all BB-UMs

The CM shall extract the participating BBs of the vehicle out of the active Vehicle Manifest. On every request to the BB-UMs in every state the CM shall observe the time between the request respectively and the responds for each associated BB-UM to verify their availability and proper operation. In case the specified time has expired, the CM shall set the life check status of the BBs accordingly.

5.1.3.4 CM provides BB Manifest

On Normal Operation a BB-UM shall download its BB Manifest from the CM periodically and compare it with its stored BB Manifest. The CM shall therefore provide the active BB Manifest to the BB-UM.

5.1.3.5 CM gathers update recommendation

After BB Manifest comparison, the BB-UM shall send the actual update recommendation to the CM. The CM shall therefore gather the update status from all participating BB-UMs and store a summary of the outcome that can be queried by command.

5.1.3.6 CM sends update request

If all participating BB-UMs have sent their update recommendation and at least an update to one BB is necessary, the CM shall send an update request to all BB-UMs. The BB-UMs decide itself if an update shall be executed. When the conditions are fulfilled, the BB-UM shall respond with update accepted respectively on time expiration update rejected.

5.1.3.7 CM gathers update accepted

After all BB-UMs have received their update request, the CM shall gather the update accepted status from the BB-UMs and provide a summary of the outcome. In case of a rejection of one or more BB-UMs the CM shall retry the update request or prompt for user interaction*). If all BB-UMs have sent their acceptance respectively turned to state Update Pending the CM shall change to state Update Pending as well.

*) Look up 5.1.2.7 manual or auto mode

5.1.3.8 CM evaluates practicable update sequence

The CM shall be in a position to optimize the update process and determine a logical and practicable update sequence automatically out of the BB dependency description of the BB Manifests. If no BB dependency description is defined, the CM shall serve the BB-UMs in the order of the incoming BB-UM update requests.

5.1.3.9 CM sends update trigger

If all concerned BB-UMs are in state Update Pending, the CM shall send an update trigger to all BB-UMs to start the update. When all BB-UMs have changed to state Update the CM shall change to state Update as well.

5.1.3.10 CM provides BB Package

After a BB-UM has received its update trigger and therefore changed to state Update, it immediately downloads the active BB Manifest and verifies, if the BB Manifest matches with the actual stored BB Manifest *). Only if the BB Manifest doesn't match the BB-UM shall download its active BB Package including firmware and parameters from the CM.

*) In this state the BB-UM has already downloaded an active BB Manifest from the CM. The initial repetition of downloading the BB Manifest is necessary, because a transition to state Update shall be possible not only from state Update Pending but as well from state Update Completed. In case of a rollback the CM has changed the active BB Manifest in the meantime.

5.1.3.11 CM gathers update result

After all concerned BB-UMs have downloaded their BB Package, the CM shall wait and gather the result of the update and installation from the BB-UMs and provide a summary of the outcome. The outcome of the installation shall be differentiated between update successful, verification failure, installation failure, no update required. If all associated BB-UMs have sent their update result to the CM, the CM shall turn to state Update Completed as well.

5.1.3.12 CM requests to restart the system

Once an update and installation are completed successfully the CM shall have the possibility to request the BB-UMs to change to state Restart to request a restart of the system. The BB-UMs shall respond with restart accepted before the restart is executed.

5.1.3.13 CM requests change to Normal Operation

Once an update and installation are completed successfully and a restart was already executed the BB-UMs shall resume to state Update Completed. The CM shall have the possibility to request the BB-UMs to change to state Normal Operation. The BB-UMs shall respond with "change to Normal Operation" accepted. If all associated BB-UMs have sent their acceptance to change to state Normal Operation, the CM shall change to state Normal Operation as well.

5.1.3.14 CM requests re-update or rollback

If the outcome of the latest update process indicates a failure on one or more BB-UMs, the CM shall have the possibility to send an update trigger to the associated BB-UMs again. Compare 5.1.3.9. In case of a rollback **) the CM shall set the previous BB Manifest to active in advance, before the update of all associated BB-UMs is triggered again.

**) The use case of a rollback of an "old" device from stock, where the original Configuration is unknown by the CM, is not part of the PoC.

5.1.3.15 CM request to cancel update

Once the update request was sent from the CM to all associated BB-UMs, the BB-UMs are waiting for the trigger from the IEC-App indicating update is permitted. The CM therefore shall have the possibility to cancel the pending update and allow the BB-UMs to return from state Update Pending to state Normal Operation.

5.1.3.16 CM changes to error state

Upon fatal error of the CM, the CM shall change to state Error and prompt for a shutdown or restart command.

5.2 Common requirements of the CM

5.2.1 Status message definitions

The CM shall gather and update the status information (see 5.2.1) of all BB-UMs and provide the status information of its own state and active Vehicle Manifest.

The status message of a BB-UM shall contain the following information. The life check status of every BB-UM shall be evaluated and set by the CM:

```
struct BB_Status_T {
    STRING          s_ManifestName;
    STRING          s_ReleaseVersion;
    STRING          s_FunctionalVersion;
    STRING          s_TypeID;
    BOOL            x_IsIntegrityValid;
    BOOL            x_IsSignatureValid;
    BOOL            x_IsRootOfTrustValid;
    BOOL            x_IsUpdateRecommeded;
    BOOL            x_IsUpdateRequested;
    BOOL            x_IsUpdateAccepted;
    BOOL            x_IsLiveCheckValid;
    BB_InstallResult_E e_BB_InstallResult;
    BB_UpdateState_E  e_BB_UpdateState;
}
```

The status message of the CM shall contain the following information:

```
struct CM_Status_T {
    STRING          s_VehicleManifestName;
    STRING          s_JsonFileName;
    STRING          s_ReleaseVersion;
    STRING          s_TypeID;
    BOOL            x_IsIntegrityValid;
    UINT16          u16_ActiveVehicleManifestIndex;
    BOOL            x_IsStarted;
    BOOL            x_IsAutoMode;
    BOOL            x_IsLogExtended;
    BOOL            x_IsIntegrityValid;
    BOOL            x_IsSignatureValid;
    BOOL            x_IsUpdateRecommeded;
    BOOL            x_IsUpdateAccepted;
    CM_DisplayMode_E e_DisplayMode;
    CM_UpdateState_E e_CM_UpdateState;
    BB_Status_T      at_BB_Status;
}
```


5.2.2 CM/ BB-UM command definitions

The interaction of the state machines of the CM and the BB-UMs is performed by commands.

5.2.2.1 Commands from BB-UM to CM

The BB-UM requests the CM for a file transfer by the following commands:

- Download BB Manifest, BB Manifest sent back (publish active BB Manifest) ΔT information outcome
- Download BB Package, Publish active BB package, ΔT information outcome

5.2.2.2 Commands from CM to BB-UM

The CM controls the BB-UM state machine by the following commands:

- Request for BB-UM status, respond with BB-UM status or timeout
- Request for update, response with BB-UM status: update accepted, denied or timeout
- Send update trigger, response with status: update trigger accepted or timeout
- Request for restart, response with BB-UM status: restart accepted, rejected or timeout
- Request for cancel/abort, response with BB-UM status: cancel accepted, rejected or timeout
- Request for turning to normal operation, response with BB-UM status: transition accepted, rejected or timeout

5.2.3 CLI command definitions

The CLI shall offer operational commands for the CM and test commands forwarded via CM to the participating BB-UMs to take effect on the state transitions of the BB-UMs manually.

5.2.3.1 CLI Operational commands

- *cm_start* start the Configuration Manager
- *cm_stop* stop the Configuration Manager
- *cm_status* list status information of Configuration Manager
- *bb_status* <all, 1..n> list status information of BB-Update Manager
- *vc_list* <all> list a summary of all available Vehicle Configurations
- *vc_list* <1..n, name> list the complete content of a specific Vehicle Configuration
- *vc_activate* <1..n> activate specific Vehicle Manifest (all others are set to deactivated)
- *vc_verify* <all, 1..n, name> verify and list verification result of Vehicle Configuration *)
- *cm_mode* <auto, manual> set mode of CM to auto or manual
- *cm_version* display name and version of the CM application
- *log_mode* <normal, extended> set the logging mode of the CM to normal or extended

- *log_list* <all, max> list content of the latest logfile, all or the latest max lines
- *display_mode* <off, normal, extended> set the display mode for the CLI to off, normal or extended

*) Recursive for all files dependent from and referenced by the Vehicle Manifest

5.2.3.2 CLI Test commands

- `ev_update_request` request update of all associated BBs. Restricted to state Normal Operation
- `ev_update_cancel` cancel update of all associated BBs. Restricted to state Update Pending to cancel an already
- `ev_update_trigger` trigger update of all associated BBs. Restricted to states Update Pending and Update Completed to trigger an already accepted update respectively repeat an update in case of failure
- `ev_restart <all, 1..n>` restricted to state Update Completed
- `ev_normal_operation <all, 1..n>` restricted to state Update Completed
- `ev_update_accepted <all, 1..n>` for the PoC to simulate the verification of the update condition executed normally from the IEC App.

5.2.4 Command response

The CM shall confirm the execution of a command from the CLI with a positive or negative responds. Example output of the CLI:

```
> vc_list 1
-- File Vehicle_TrackRunner_1_1_1.json
----- Systems CCS-OB
----- BB DAS
----- File DAS_v1.1.0.json
----- Archive xCPU94x_vPLC_Firmware_DAS_V1.1.0.tar.gz
----- Archive xCPU94x_IEC_Application_DAS_V1.1.0.zip
----- BB ETP
----- File ETP_v1.1.9.json
----- Archive xCPU94x_vPLC_Firmware_SERP_V1.1.9.tar.gz
----- Archive xCPU94x_IEC_Application_SERP_V1.1.9.zip

> vc_verify 1
--Please compare the sha256 hash:
---959cb8ee3218cbcb6144590e07c638fedacb29094b3a695e7a471cdac6fef602d
--of this file:
---/home/gfl/ConfMgr/repo/vehicle1/v1/Vehicle_TrackRunner_System-neu.json
--with an externally obtained hash of the file.

--2024-03-14T16:49:37: Signature of System 0 ref. BB 0 is matching
--2024-03-14T16:49:37: Signature of System 0 ref. BB 0 is matching on lower layer too
--2024-03-14T16:49:37: Signature of System 1 ref. BB 0 is matching
--2024-03-14T16:49:37: Signature of System 1 ref. BB 0 is matching on lower layer too
```

5.2.5 Logging of events

On every event listed in the table below the essential information of the event shall be written into a logfile "EventLog_n.txt". In log mode "normal" only a subset of the events shall be considered according of the table below. If the logfile reaches a defined limit of entries a subsequent logfile shall be created with the index n incremented.

Timestamp	Category	Source	Event description
23.11.2023 17:54	INFO	BB-UM	Installation of new firmware successful

23.11.2023 18:21 / INFO / CM / Update of Vehicle Manifest successful
23.11.2023 19:36 / INFO / CLI / Configuration Manager stopped by command

The categories INFO, WARN and ERR shall be distinguished. Possible sources shall be BB-UM, CM and CLI. The content of the logfile shall be displayed by command.

5.2.6 Displaying of status information

On every event listed in the following table the essential status information shall be displayed to the console. In display mode “normal” only a subset of the events shall be considered according of the table below. In display mode “off” the information shall only be displayed by command.

Event	Normal mode	Extended mode
On startup of the CM:	The CM shall prompt/log name, state and version “CM is operating in start/stop mode vx.x.x”.	The CM shall prompt/log name, state and version “CM is operating in start/stop mode vx.x.x”.
On status information is requested from the CM:	-	The CM shall display/log the action.
On initial status information of a BB-UM is available:	-	The CM shall display/log the status result of the BB-UM.
On providing BB Manifests to the BB-UMs:	-	The CM shall display/log the action initially of each BB-UM with name and identifier that starts the download.
On download recommendation is available of all associated BB-UMs:	The CM shall display/log a summary of the download recommendation.	The CM shall display/log a summary of the download recommendation and additionally list the download recommendation of each associated BB-UM.
On update is requested from the CM:	-	The CM shall display/log the action.
On update acceptance is available of all associated BB-UMs:	The CM shall display/log a summary of the update acceptance.	The CM shall display/log a summary of the update acceptance and additionally list the acceptance of each associated BB-UM.
On update trigger is sent to all BB-UMs:	-	The CM shall display/log the action.
On update cancel is sent to all BB-UMs:	-	The CM shall display/log the action.
On providing BB Package to the BB-UMs:	-	The CM shall display/log the action initially of each BB-UM with name and identifier that starts the download.
On update result is available of all associated BB-UMs:	The CM shall display/log a summary of the update result.	The CM shall display/log a summary of the update result and additionally list the result of each associated BB-UM.
On restart request of the CM is sent to all associated BB-UMs:	The CM shall display/log the action.	The CM shall display/log the action.
On change to Normal Operation is sent to all associated BB-UMs:	The CM shall display/log the action.	The CM shall display/log the action.
On error request is received:	The CM shall display/log the action.	The CM shall display/log the action.
On exception in auto mode is occurred:	The CM shall display/log the action.	The CM shall display/log the action.

On command received from CLI	The CM shall display/log the confirmation of the command	The CM shall display/log the confirmation of the command
On parsing a Vehicle Manifest	The CM shall only display/log the final result of the parsing procedure and in case of an error the error messages.	<p>The CM shall display all parsing events.</p> <p>The CM log only the final result of the parsing procedure and in case of an error the error messages.</p>

6. Manifest requirement specification

The following chapters describe the definition and requirements of a Manifest and its depending items. The goal is the formulation of a generic approach on the vehicle, system and device level. This approach thus deviates from the original specification [R3] in chapter 5.

6.1 Requirements

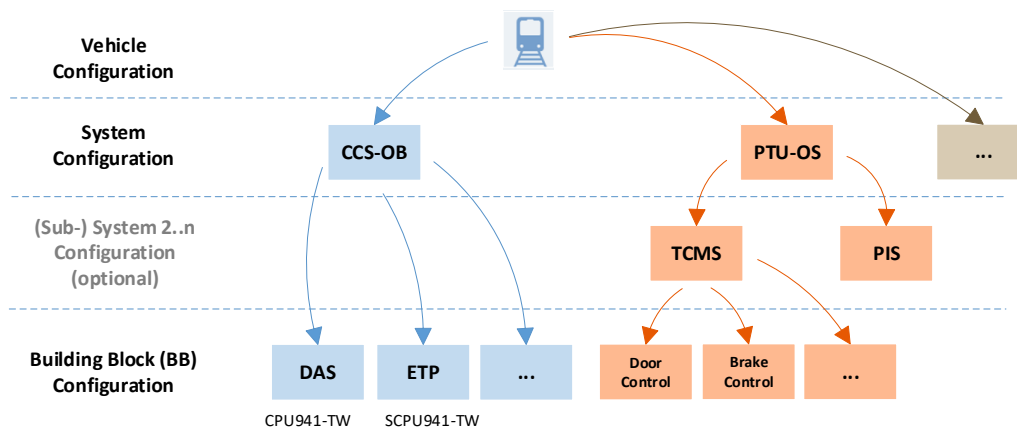
6.1.1 File format of a Manifest

Since the Manifest shall be written in a human readable, standardized lightweight data-interchange format, the Manifest shall be written preferably in the JSON file format.

6.1.2 Pattern of a Manifest

Since each level and instance shall have its own Configuration respectively Manifest, the individual Manifests in the same file must be combinable analogous to a hierarchal tree pattern or referenceable in case of distribution to different Manifest files. The granularity of the breakdown into different Manifest files shall be freely selectable with the goal of being able to realize an overall structure that is as clear and as readable as possible. The optional grouping into system and sub-systems is therefore only a structural aid and has no further function. The Manifest for the vehicle, system and Building Blocks (device) are mandatory – the optional grouping into one or more system levels is optional and depends on the complexity and logical respectively physical relations of the whole system.

Figure 6: Hierarchical levels of the different Configurations / Manifests



6.1.3 Mandatory and optional elements of the Manifest

To cover diverse aspects on the different configuration levels with a generic approach, mandatory and optional elements shall be configurable.

6.1.4 Support of referenced files

Since different Manifests shall be chainable and also referenceable in case of distribution to different Manifest files, handling of the different files shall be supported. Moreover within the

Manifest structure it shall also be possible to reference to other files like software and parameter packages.

6.1.5 Integrity

The integrity of the Manifest with all its referenced and nested Manifests and items shall be checked by the parser.

6.1.6 Identification and authentication

With the elements of the Manifest it shall be possible to identify the target of a file and verify its authenticity (source supplier, integrator, distributor). To support the identification and authentication process, all entities involved in the Configuration Management shall use unique identifiers. Cryptographic techniques shall therefore be applied. In conclusion, a Configuration shall only be activated after a positive verification of the authenticity of its source, ascertainment that the source is a trusted Configuration Management entity, and that the configuration is targeted to the correct receiving entity.

6.1.7 Compatibility and versions of different BBs

To support the integrator in the process of building the different Manifests, a functional version $x.y.z$ shall be used to validate the compatibility of the Building Blocks to be integrated. The functional version shall consist of a major (x), minor (y) and a patch (z) index. If BB_1 depends on BB_2 , they shall be compatible, if their major indices are equal $x_1 = x_2$ and their minor indices fulfill the equation $y_2 \geq y_1$. The patch indices z_1 and z_2 can take any integer $z \in \mathbb{N}_0$ and have no relevance for the compatibility verification.

6.1.8 BB dependencies and update hierarchy

The Building Blocks can have dependencies to other Building Blocks. The definition of such dependencies shall be represented in the BB Manifest. The defined dependencies shall contain the required version. The CM shall therefore be able to optimize the update process and determine a logical and practicable update sequence automatically. The dependency definitions shall be mandatory.

6.1.9 Parser capabilities

The parser for the Manifests and referenced items shall fulfill the following requirements.

6.1.9.1 Reading information

The parser shall read and extract the corresponding information of each item out of the Manifest with respect to the used file format.

6.1.9.2 Retrieving file paths

Since a Manifest references other Manifests the parser shall retrieve the corresponding file paths and names and continue parsing within the referenced Manifest according to the nested Manifest structure.

6.1.9.3 Checking syntax and semantic

The parser shall check the syntax and semantics of the Manifests with respect to all nesting levels. In case of an error the parser shall give a helpful hint to correct the syntax or semantic of the Manifests accordingly.

6.1.9.4 Checking integrity

The parser shall check the integrity of all referenced Manifests and referenced Archives with respect to all nesting levels. In case of an error the parser shall give a helpful hint to correct the missing or incomplete part of the Manifests accordingly.

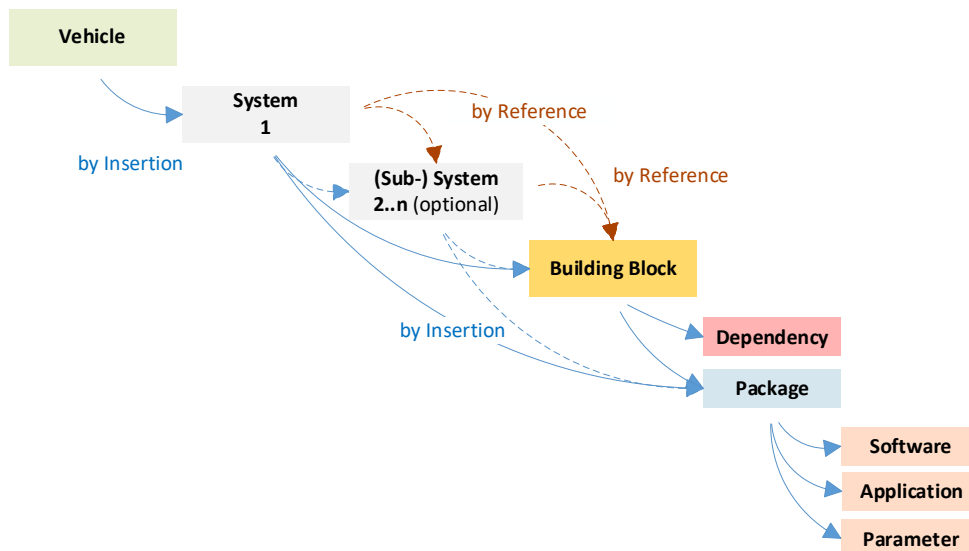
6.1.10 Config Items and Config Types in the Manifest structure

A Manifest consists of Config Items associated to different Config Types. The following two figures illustrate the dependency respectively the nesting possibilities of different Config Types.

6.1.10.1 Dependency of Config Types

The dotted lines indicate optional or alternative dependencies. If the Manifests are spitted in different files they shall be included by reference. The Vehicle Manifest contains Items of type System, Building Block and Package. The BB Manifests contains Items of type Dependency and Package. The Items of type Package reference to Items of type Software or Parameter.

Figure 7: Dependency of Config Types



6.1.10.2 Nesting possibility of Config Types

Package Items shall be allowed to be integrated in the Vehicle, System and Building Block Items. However, the Package Items for Vehicle and System Items shall only contain Parameter Items – Software Items shall only be allowed in combination with Building Block Items.

Figure 8: Nesting possibilities of Config Types

Config Type \ Config Item	Vehicle	System	Building Block	Dependency	Package	Software	Parameter	Application
Vehicle	x	x	x	x	x	x	x	x
System	✓	✓	x	x	x	x	x	x
Building Block	x	✓	x	x	x	x	x	x
Dependency	x	x	✓	x	x	x	x	x
Package	x	✓	✓	x	x	x	x	x
Software	x	x	x	x	✓	x	x	x
Parameter	x	x	x	x	✓	x	x	x
Application	x	x	x	x	✓	x	x	x

6.1.11 Config Items and its Elements

Each Config Item has a set of Elements and the Element Config Type specifies the type of the Config Item. Some of the Elements shall be used for all Config Types and some of them are type specific. Some of them shall be mandatory (M), some of them are optional (O) and some of them unused (-) for the corresponding Config Item. If an Element has a default value, the explicit usage of the Element shall be optional.

Figure 9: Config Items and Elements of a Manifest

The following chart defines the usage of the Elements for each Config Item.

Config Item \ Elements	Vehicle	System 1 mandatory	(Sub-) System 2..n optional	Building Block	Depend- ency	Pack- age	Soft- ware	Param- eter	Appli- cation
✓ Config Type String specifies the type of a Config Item	M	M	M	M	M	M	M	M	M
✓ Type ID String with UUID format	M	M	M	M	M	M	M	M	M
✓ Config ID String with UUID format	M	M	M	M	-	-	-	-	-
✓ Name String	O	O	O	O	O	O	O	O	O
✓ Description String	O	O	O	O	O	O	O	O	O
✓ Supplier String	O	O	O	O	O	O	O	O	O
✓ Metadata String	O	O	O	O	O	O	O	O	O
✓ Functional Version String with x.y.z format	-	-	-	M	M	-	-	-	-
✓ Release Version String with x.y.z format	M	M	M	M	-	M	M	M	M
✓ Manifest String with relative path to a json-file	-	-	-	-	-	-	-	-	-
✓ Archive String with relative path to a tar.gz-file	-	-	-	-	-	-	M	M	M
✓ Network IDs List of Strings with IPv4 format	-	O	O	M	-	-	-	-	-
✓ Systems List of Items of type System	M	O	O	-	-	-	-	-	-
✓ Building Blocks List of Items of type BB	-	M	M	-	-	-	-	-	-
✓ Packages List of Items of type Software or Parameter	-	O	O	M	-	-	-	-	-
✓ Reference Triple with three Elements	-	M	M	-	-	-	-	-	-
✓ Dependencies List of Items of type Dependency	-	-	-	M	-	-	-	-	-
✓ Hash Type String SHA256 default	-	-	-	-	-	-	O	O	O
✓ Hash String coded according Hash Type	-	-	-	-	-	-	M	M	M
✓ Signature Type String PKCS#1 default	O	O	O	O	-	-	-	-	-
✓ Signature String coded according Signature Type	M	M	M	M	-	-	-	-	-

6.1.12 Explanation of the Elements

6.1.12.1 Config Type

By introducing the Element Config Type, type-related differences regarding type specific, optional and mandatory Elements shall be checked in the Manifest. The format of this Element shall be a string and only defined Config Types shall be allowed by the parser. The Config Type shall be defined as an ENUM to specify the allowed values. Config Types are:

- | | | |
|--|---|--|
| ■ Vehicle | ■ Dependency | ■ Software |
| ■ System | ■ Package | ■ Parameter |
| ■ Building Block | | ■ Application |

6.1.12.2 Type ID and Config ID

Each Config Type shall have an Element Type ID and each Manifest in a separate JSON file shall have an Element Config ID. For a Config or Type ID an Universally Unique Identifier (UUID) shall be used. The composition of the UUID is not yet specified but for a Building Block Configuration may be based on a combination of a datetime value and a part of the MAC address of the device – and for a Vehicle Configuration generated by hashing a combination of elements in the Manifest. For the Network Identifier a list of IP-addresses (IPv4) shall be used. The format of the identifiers shall be a string and the parser shall check the specific UUID respectively IP format.

6.1.12.3 Name, Description, Supplier and Metadata

Some additional Metadata shall be used to describe Manifest specific information. Essential metadata are Name, Description and Supplier. The format of these Elements shall be a string.

6.1.12.4 Release and Functional Version

The Element Release Version is provided from a supplier for a Building Block, Package, Firmware or Parameter Item or from an Integrator for a Vehicle or System Manifest. The Element Functional Version instead shall guarantee the compatibility between different Building Blocks of different suppliers. The type of the version shall be a x.y.z coded string (see 6.1.7).

6.1.12.5 Manifest

The Element Manifest is part of the Vehicle or System Config Item and shall reference to a dependent Manifest in a separate file with the extension *json*. The used path shall be relative and the format shall be a path coded string `<./path/name.json>`. The Element is optional because nested Manifests shall also be directly includable in their parent Vehicle or System Manifest.

6.1.12.6 Archive

The Element Archive is part of the Software or Parameter Config Item and shall reference to a separate file with the extension *zip* or *tar.gz* including firmware or parameter settings. For the PoC only *tar.gz* shall be supported. The used path shall be relative and the format shall be a path coded string `<./path/name.tar.gz>`.

6.1.12.7 Network IDs

The Element Network IDs shall be a list of strings that define the used and allowed IPv4 addresses on the local network. For each BB the definition shall be mandatory but it shall be also definable on level of the Vehicle or System Manifest. If no definitions are made on BB Manifest level the addresses shall be derived from the parent levels.

6.1.12.8 Systems

An Element Systems shall be a possibility to combine any number of Building Blocks to logical coherent sub-systems, e.g. BB “DAS” and BB “ERP” are grouped to system “CCS-OB” (see Figure 6). Only the initial System in the Vehicle Manifest is mandatory – further nesting is optional, e.g. the BB “Door Control” and BB “Brake Control” are grouped to sub-system “TCMS” (see Figure 6). The optional grouping is only a structural aid and has no further function. Nested Systems of any number shall be possible as composition or reference to another System Manifest.

6.1.12.9 Building Blocks

The Element Building Blocks shall be used within a Vehicle or System Manifest and include all associated BB Manifests either as reference to a file or directly inserted with its Config Items and Elements. Building Blocks shall contain only Dependency and Package Items and within the Package Items Software and Parameter Items shall be allowed.

6.1.12.10 Packages

The Element Packages shall be used within a Vehicle, System or BB Manifest and include all associated Software and Parameter Items. However Software Items are only allowed in Packages used by Building Blocks.

6.1.12.11 Reference

The Element Reference shall be used to refer to nested (Sub-)System- or BB-Manifests that are stored in separate files. The Element Reference shall be a Triple containing three Elements: Name, Signature and Manifest. While parsing a Manifest a Reference shall be completely replaced by the referenced Manifest.

6.1.12.12 Dependencies

The Element Dependencies shall be used to link dependent BBs together. The Type ID of the dependent BB and the Functional Version shall be assigned to each Dependency to allow automatically executed compatibility checks.

6.1.12.13 Hash Type

The default value for the Hash Type shall be SHA256. Other Hash Types are not supported for the PoC.

6.1.12.14 Hash

A Hash shall be assigned to each Archive referenced within the Software or Parameter Item of a Package. Hashing shall be done according to the Hash Type. It shall be used to verify the integrity and identity of the referenced Archive and to detect any tampering and modifications.

6.1.12.15 Signature Type

The default value for the Signature Type shall be PKCS#1. Other Signature Types are not supported for the PoC.

6.1.12.16 Signature

A cryptographic algorithm shall be used to sign the hashed content of every Manifest. If a BB Manifest is included within a Vehicle or System Manifest by composition (and not by reference) the corresponding Building Block shall additionally be secured by a signature. The signature shall be used for integrity-, identity- and authentication checks. This allows to detect any tampering, modifications and to verify the root of trust.

6.1.13 Example JSON files

The examples for a JSON file for the Vehicle Configuration with a System named CCS-OB and two Building Blocks DAS and ERP lean on the overview described in Figure 1 for the PoC. The System is composed within the Vehicle Manifest and not excluded to a separate file. The Building Blocks BB DAS and BB ETP are configured reference.

6.1.13.1 Example Manifest of a Vehicle Configuration

```
{
  "Config Type" : "Vehicle",
  "Config ID" : "8259d77b-e4e7-4261-8025-fa92c3a0a795",
  "Type ID" : "6c4d65f4-2572-11ee-be56-0242ac120002",
  "Name" : "Track Runner 2",
  "Supplier" : "Acme Corp.",
  "Release Version" : "1.1.1",
  "Systems" : [
    {
      "Name" : "CCS-OB",
      "Config Type" : "System",
      "Type ID" : "6c4d65f4-2572-11ee-be56-0242ac1102345",
      "Description" : "Control-Command and Signaling On Board",
      "Release Version" : "1.1.0",
      "Building Blocks" : [
        {
          "Reference" : {
            "Name" : "DAS",
            "Signature" : "6f049c1ec76f49ac82c1474084421abc4661be1978cfdab1c095...",
            "Manifest" : "./System/CCS-OB/BBs/DAS_v1.1.0.json"
          }
        },
        {
          "Reference" : {
            "Name" : "ETP",
            "Signature" : "6f049c1ec76f49ac82c1474084421abc4661be1978cfdab1c093...",
            "Manifest" : "./System/CCS-OB/BBs/ETP_v2.1.0.json"
          }
        }
      ]
    },
    {
      "Config Type" : "Parameter",
      "Type ID" : "6c4d65f4-2572-11ee-be56-0242ac120003",
      "Description" : "Vehicle fe0001 Parameter",
      "Release Version" : "1.1.9",
      "Archive" : "./Vehicle_fe0001_Parameter_v1.1.9.zip",
      "Signature" : "9785eb76c2ea023ca2f4904b9cb5672d62dc78b2acf28d702827376a2"
    }
  ],
  "Packages" : [
    {
      "Config Type" : "Parameter",
      "Type ID" : "6c4d65f4-2572-11ee-be56-0242ac120003",
      "Description" : "Vehicle fe0001 Parameter",
      "Release Version" : "1.1.9",
      "Archive" : "./Vehicle_fe0001_Parameter_v1.1.9.zip",
      "Signature" : "9785eb76c2ea023ca2f4904b9cb5672d62dc78b2acf28d702827376a2"
    }
  ],
  "Signature" : "07d385472a15456fb12687bf9d9dbbfc153de81ba6b3b0195256cf5b5ac8"
}
```

6.1.13.2 Example Manifest of a Building Block

The referenced BB DAS Manifest "DAS_v1.1.0.json" from the Vehicle Manifest example:

```
{
  "Config Type" : "Building Block",
  "Config ID" : "8259d77b-e4e7-4261-8025-fa92c3a0a795",
  "Type ID" : "6fcf44f5-f9a5-4da1-93e9-8c27ffe02d36",
  "Name" : "DAS",
  "Release Version" : "3.0.900",
  "Functional Version" : "1.1.0",
  "Supplier" : "Octan",
  "Network IDs" : [
    "192.168.0.201",
    "192.168.0.1"
  ]
}
```

```

],
"Dependencies" : [
{
  "Config Type" : "Dependency",
  "Type ID" : "6e8fa7b0-7fe2-4d08-9101-b22f086f0787",
  "Name" : "ETP",
  "Functional Version" : "2.1.0"
}
],
"Packages" : [
{
  "Type ID" : "fce031f8-7462-4ad1-b320-7fcba3329baa",
  "Config Type" : "Software",
  "Name" : "MOS SCPU941 (vPLC)",
  "Release Version" : "3.0.900",
  "Signature" : "81dc096b6ab943bfa6c03961fed1c8c72745cff5382ddec...",
  "Archive" : "./xCPU94x/MOS_xCPU941_V3.0.17.tar.gz"
},
{
  "Type ID" : "e5c2c137-ee35-4044-a41b-42169e9d2b22",
  "Config Type" : "Application",
  "Name" : "IEC App DAS",
  "Release Version" : "1.1.0",
  "Signature" : "fb6aefb7562f4817a865e5f2e9fcfb99da2e1f15c774aec123...",
  "Archive" : "./IEC_App/IEC_App_DAS_V1.1.0/res.tar.gz"
}
]
"Signature" : "6f049c1ec76f49ac82c1474084421abc4661be1978cfdab1c095..."
}

```

The referenced BB ETP Manifest "ETP_v2.1.0.json" from the Vehicle Manifest example:

```

{
  "Config Type" : "Building Block",
  "Config ID" : "8ba8d7a2-5f09-46bd-adf7-ccace4400d41",
  "Type ID" : "9f671cb3-c213-4c71-a15d-0e25fb0ccab1",
  "Name" : "ETP",
  "Description" : "This is the all new, highly performant ETP implementation",
  "Supplier" : "Hulesch & Quenzel",
  "Release Version" : "3.0.900",
  "Functional Version" : "2.1.0",
  "Network IDs" : [
    "192.168.0.211",
    "192.168.0.2"
  ],
  "Dependencies" : [
    {
      "Type ID" : "6e8fa7b0-7fe2-4d08-9101-b22f086f0787",
      "Config Type" : "Dependency",
      "Name" : "DAS",
      "Functional Version" : "1.1.0"
    }
  ],
  "Packages" : [
    {
      "Type ID" : "fce031f8-7462-4ad1-b320-7fcba3329baa",
      "Config Type" : "Software",
      "Name" : "MOS SCPU941 (vPLC)",
      "Release Version" : "3.0.900",
      "Signature" : "a22d3d8621ae4221be96e915cde4c1f2fc055ccf1c4f40b3a033e16057baddaf",
      "Archive" : "./xCPU94x/MOS_xCPU941_V3.0.17.tar.gz"
    },
    {
      "Type ID" : "e5c2c137-ee35-4044-a41b-42169e9d2b22",
      "Config Type" : "Application",
      "Name" : "IEC App ETP",
      "Release Version" : "1.2.0",
      "Signature" : "70e4fc8ed40742f4aabef9767f19a67f605689649b814123a78a0198e3dfa16b",
      "Archive" : "./IEC_App/IEC_App_ETP_V1.2.0/res.tar.gz"
    }
  ],
  "Signature" : "6f049c1ec76f49ac82c1474084421abc4661be1978cfdab1c093..."
}

```

6.1.13.3 Example Vehicle Manifest with nested Systems

The following example illustrates a Vehicle Manifest using Sub-Systems according Figure 6. The example is incomplete and not all mandatory Elements are listed to illustrate the nesting of Systems in a compact way:

```
{
  "Config Type" : "Vehicle",
  "Name" : "Track Runner 3",
  "Release Version" : "2.2.0",
  "Systems" : [
    {
      "Name" : "CCS-OB",
      "Config Type" : "System",
      "Description" : "Element 1 of System",
      "Building Blocks" : [
        {
          "Reference" : {
            "Name" : "DAS",
            "Signature" : "6f049c1ec76f49ac82c1474084421abc4661be1978cfdab1c093...",
            "Manifest" : "./System/CCS-OB/BBs/DAS_v1.1.0.json"
          }
        },
        {
          "Reference" : {
            "Name" : "ETP",
            "Signature" : "6f049c1ec76f49ac82c1474084421abc4661be1978cfdab1c093...",
            "Manifest" : "./System/CCS-OB/BBs/ETP_v1.1.9.json"
          }
        }
      ]
    },
    {
      "Name" : "PTU-OS",
      "Config Type" : "System",
      "Description" : "Element 2 of System with 2 Sub-Systems",
      "Systems" : [
        {
          "Name" : "TCMS",
          "Config Type" : "System",
          "Description" : "Element 1 of Sub-System PTU-OS",
          "Building Blocks" : [
            {
              "Reference" : {
                "Name" : "Door Control",
                "Signature" : "6f049c1ec76f49ac82c1474084421ab661be1978cfdab1c093...",
                "Manifest" : "./System/PTU-OS/TCMS/BBs/DCL_v1.2.2.json"
              }
            },
            {
              "Reference" : {
                "Name" : "Brake Control",
                "Signature" : "6f049c1ec76f49ac82c1474084421abbe1978cfdab1c093...",
                "Manifest" : "./System/PTU-OS/TCMS/BBs/BCL_v1.2.5.json"
              }
            }
          ]
        },
        {
          "Name" : "PIS",
          "Config Type" : "System",
          "Description" : "Element 2 of Sub-System PTU-OS",
          "Building Blocks" : [
            {
              "Reference" : {
                "Name" : "...",
                "Signature" : "...",
                "Manifest" : "..."
              }
            }
          ]
        }
      ]
    }
  ],
  "Signature" : "07d385472a15456fb12687bf9d9dbbfc153de81ba616a44b3b0195256cf5b5ac8"
}
```

7. Scope of functionality and restrictions for the PoC

The following restrictions compared to the PoC specification are made in the interest of a shorter implementation time and a reduction in effort:

- Building Block Manifests will only be included as reference and not in the document of the upper level.
- All structuring Items will always be marked as arrays even if there is only one element.
- Hashes will be restricted to SHA256.
- Nested systems will not be supported in the PoC parser.
- The dependencies within a BB Manifest will not be verified by the CM. Therefore the declaration of Functional Versions has no effect.
- A detailed implementation design description is not presented. Instead, all essential refinements are incorporated into the specification during implementation phase. Only a few software specific specialties are mentioned in chapter 8.
- The communication between the CM and the BB-UMs is not yet secure.
- Exchanging of software components for the CPUs are restricted to MOS (vPLC) and IEC applications.
- By writing own Manifests the *Release Version* of the MOS (vPLC) must correspond with the *Release Version* of a BB Manifest. In contrast the *Release Version* of the IEC application can be selected independently.
- With the SBB PoC test system two MOS versions 3.0.17 and 3.0.18 are delivered. Other commercial MOS versions should not be used since they have no BB-UM integrated. In contrast own IEC Applications can be build, integrated in Manifests and uploaded via CM to the BB-UMs.
- Although three independent vPLCs can be operated on a xCPU941, only vPLC1 is supported for the PoC.
- The elements Network ID, Config ID and Type ID may be set in the JSON files but haven't any effect.
- For the sake of simplicity, the BBs are addressed by the CM for the PoC via the element Name. This means that the name in the VC that references a BB must match the name of the associated BB manifest.
- Only the Release Version of the BB Manifest is used as the basis for a decision on an update recommendation.



- The system time of the CM is not set and synchronized automatically. It may be set for the by using the following commands on the CM:

```
sudo date -s 2024-04-25  
sudo date -s 11:00
```


8. Implementation

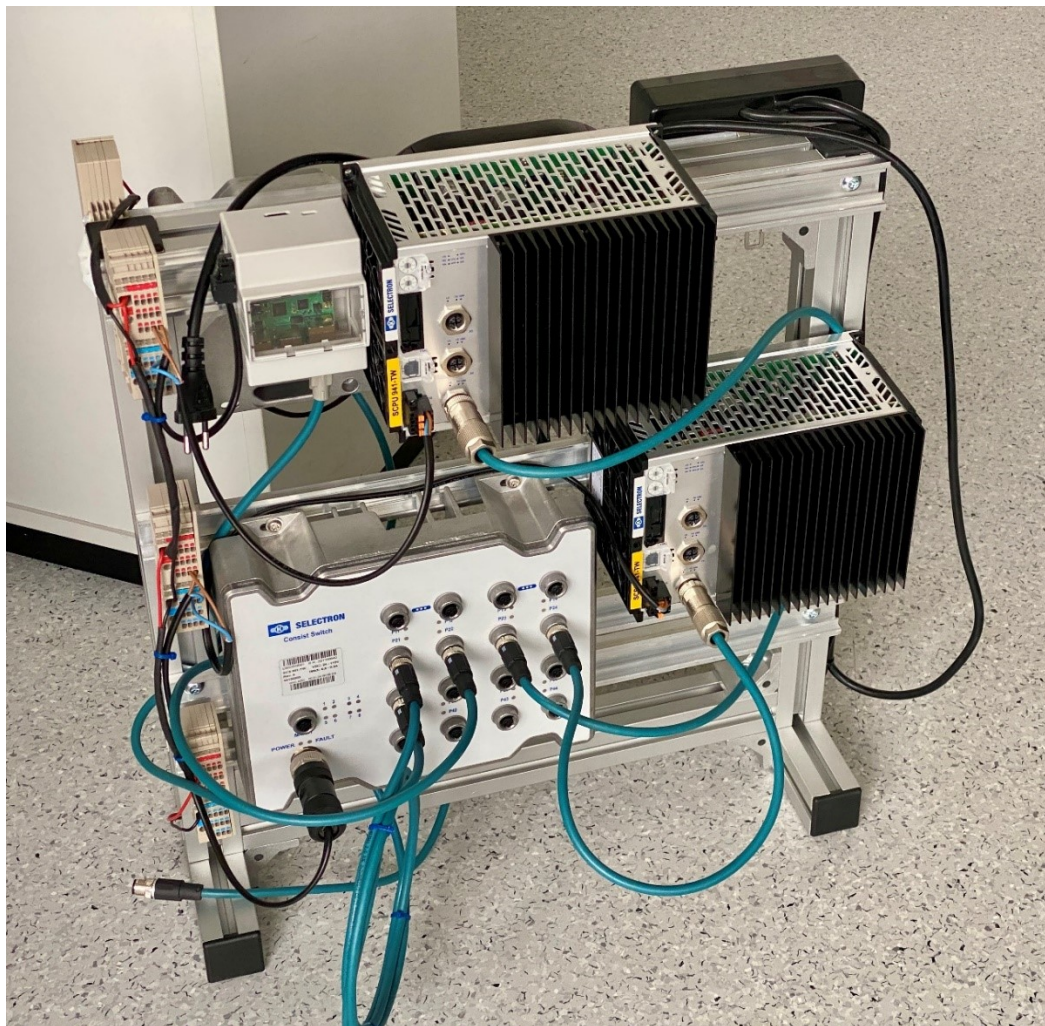
The chapter implementation is not explicitly documented. Design decisions how the specification is realized are just listed instead.

8.1 PoC Test System

The Test System for the PoC consists on the front side of two SCPU941, a Raspberry Pi, a Selectron Switch and necessary accessories and cables to connect the components. A Power Supply 24V, a Selectron Security Gateway and a Power-On Button is additionally located at the backside.

We recommend adding one SDDE301-TF, one SDOT301-TF, one SDIT301-TF and one BTM301-T to the test system for each SCPU941. The additional components can be used to effectively demonstrate the exchange of IEC apps via CM and a maintenance state can be simulated easily.

Figure 10: Components of the PoC Test System



8.2 Design decisions

8.2.1 Parser

For the JSON file handling the open source parser “parson” is used and extended.

8.2.2 Web Server

For the communication between the CM and BB-UMs a web server is realized using nginx. The file transfer is implemented by using libcurl.

8.2.3 Intermediate test environment

For implementation and test purposes of the BB-UMs an intermediate test environment is established using a Linux Ubuntu and Hyper V.

8.2.4 Obsolete CLI commands

To view the content of a log file, the log file itself must be opened. The command *log_list* is therefore not supported.

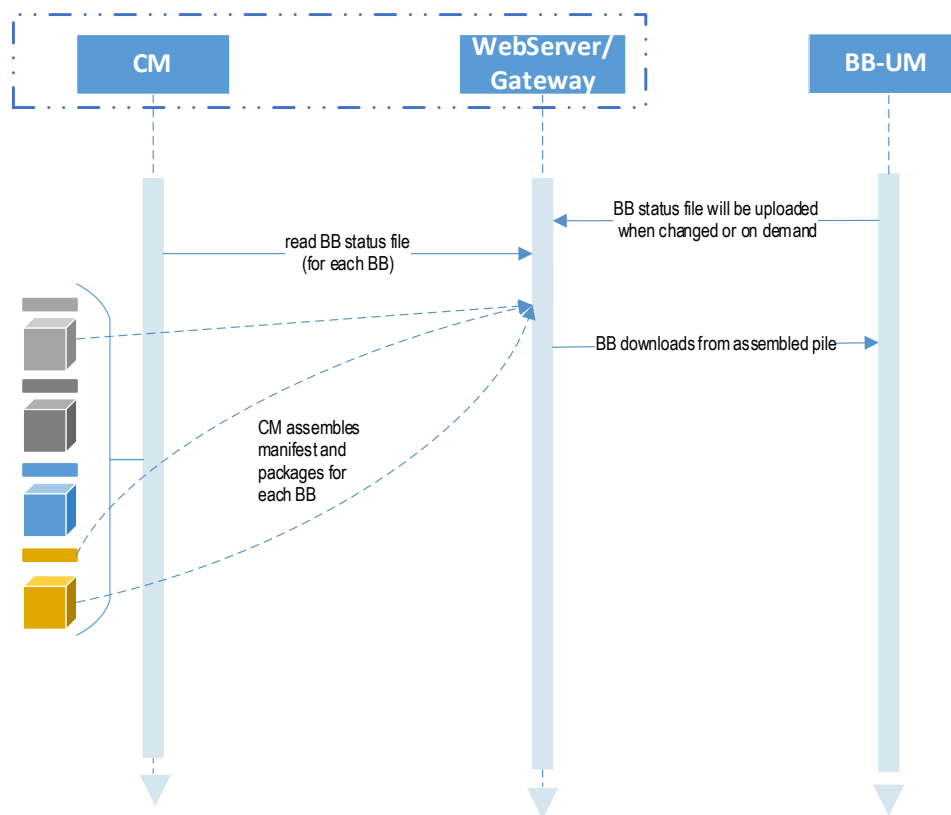
There are two modes for logging and the output on the screen: normal and extended. The *log_mode* command can be used to change the logging and display mode at the same time. This means that what is visualized is always logged. The *display_mode* command is therefore obsolete.

8.3 Implementation details

8.3.1 CM/BB-UM file handling

The exchange of files between the CM and the BB-UMs via the web server is illustrated in the following sequence.

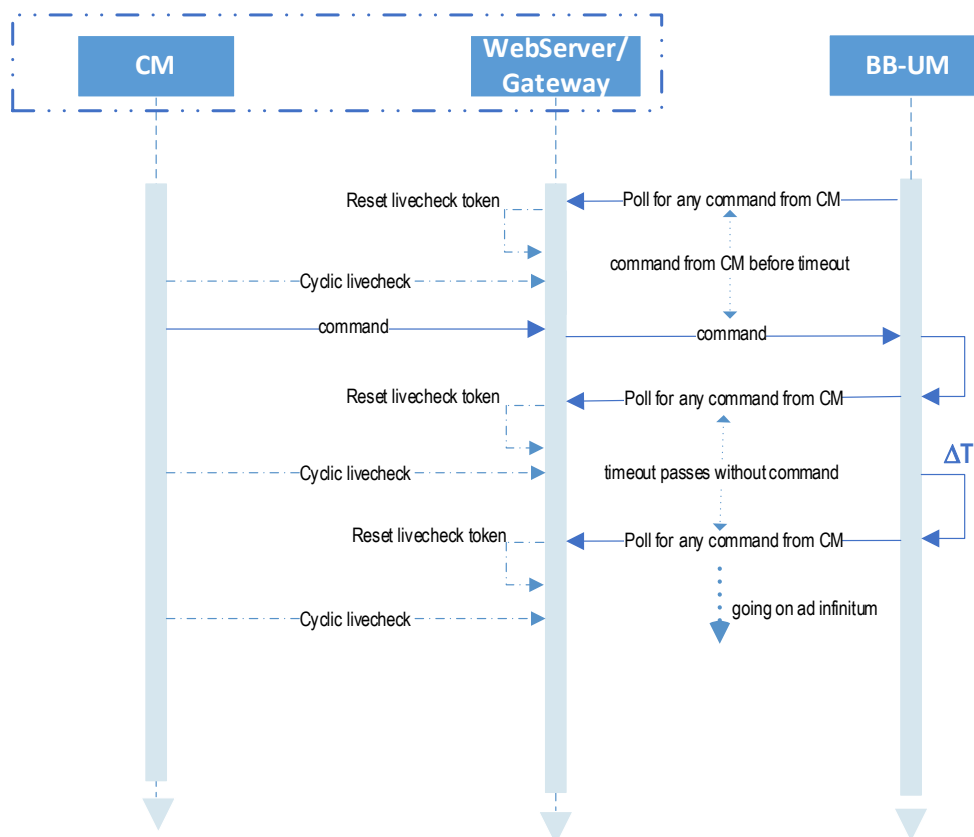
Figure 11: Sequence of CM/BB-UM file handling



8.3.2 Long Polling mechanism

Long polling is used on a web server to enable real-time communication between the server and the client without the client having to constantly send requests. The client keeps a connection to the server open and sends a request. The server keeps this request open and waits until new data is available or a time limit is reached. When new data is available, the server sends a response to the client and the process starts again. This enables efficient bidirectional communication in real time, as required for chat applications or updates in social networks, for example.

Figure 12: Sequence of the Long Polling mechanism



8.3.3 Package Config Types

During the realization of the PoC, it became apparent that not only do software and parameters need to be fundamentally differentiated within a Package, but that it also makes sense to describe software more precisely as soon as download and installation have to use different interfaces. For example, software for the PoC is associated with the operating software for the device, which usually requires the system to be restarted after an update. In contrast, “Application” software is now associated with software that only needs to be stopped, replaced and restarted. In other words, application software does not require a restart of the whole CPU. For that purposes the Config Type Application used within a Package was introduced.